

IRAF/PyRAF 講習会

(撮像データ解析編)

改訂版 (最終更新:2016 Mar 4)

2016 FEB 16 – 17 @国立天文台・天文データセンター

磯貝 瑞希@国立天文台・天文データセンター

はじめに(1): 想定受講生

本テキストが想定する受講生:

1. IRAFを対話的に使用した経験がある。
2. 少なくとも一つのプログラム言語でプログラムを書いた経験がある。

この2つの経験がない場合には、このテキストに目を通す前に、その経験を積むことを強く推奨する。

はじめに(2): 動作確認環境

本テキストのサンプルスクリプトの動作確認は、以下の環境で実施。

OS: CentOS 6.7 (64bit)

IRAF: 2.16.1 64bit

PyRAF: 2.10

Python: 2.11

本講習会の目標:

- ・CLスクリプトの基礎を学び、簡単なCLスクリプトが書けるようになる。
- ・PyRAFが使えるようになる。
- ・PyRAFを使ってデータ処理を行うPythonスクリプトが書けるようになる。

CLスクリプト実習では、複数の画像を一定の時間間隔で表示するソフトを、Pythonスクリプト実習では、撮像データの一次処理や、画像マッチング、アパーチャ測光、比較測光を行うソフトを作成する。

目次:

0. 端末の使い方
1. CLスクリプトの基礎
2. CLスクリプト実習
3. PyRAFについて
4. Python言語の基礎
5. Pythonスクリプト実習

0. 端末の使い方

0. 端末の使い方(1)

ログイン:

アカウント: `studentXX`

初期パスワード:

Xの起動方法:

ログイン後、「`startx`」を実行。

0. 端末の使い方(2)

ターミナルの開き方:

何もない場所でマウスを右クリック

→「Open in Terminal」を選択。

(~/Desktopにいるので、cdを実行してホームディレクトリに移動する)

テキストエディタ:

vi, emacs, gedit, nano

→ 使い慣れたものをどうぞ。(講師は主にemacsを使います)

日本語入力の切り替え: 「半角/全角キー」

0. 端末の使い方(3)

IRAFの起動まで:

- ・ ターミナル上で「iraf」ディレクトリを作成する:
`mkdir ~/iraf`
- ・ irafディレクトリに移動してから「mkiraf」を実行する:
`cd ~/iraf; mkiraf` (terminal typeは「xgterm」でOK)
- ・ login.clをテキストエディタで開き、set stdimageの行を以下のように編集する:
(2048 x 2048 pixelsの画像全体を表示するため)
`#set stdimage = imt800`
→ `set stdimage = imt2048` (行頭の#を削除し、imt800をimt2048に)
- ・ 「xgterm -sb &」を実行し、起動したxgterm上で「cl」を実行する。

実習資料

ホームディレクトリ下に、実習用の観測データ、サンプルスクリプトを用意。

~/pylec/

data/{100918, 100925, 100929}: 実習用観測データ (100929はフラット用)

data.bkup dataのバックアップディレクトリ (ダウンロード版では削除)

samples/

clscripts/ サンプルCLスクリプト

pyscripts/ サンプルPythonスクリプト

memo Pythonスクリプト実行メモ

clscripts/ 受講生用CLスクリプト置き場(空ディレクトリ)

pyscripts/ 受講生用Pythonスクリプト置き場(空ディレクトリ)

results/ver{1,2}/{100918,100925}/outphot.dat 比較測光結果

(ver2:改良版での結果)

doc/pylec.pdf このテキストのPDFファイル (ダウンロード版では削除)

1. CLスクリプトの基礎

CLスクリプトとは？

- ・IRAFでプログラムを書くためのスクリプト言語(コンパイル不要)。
- ・文法はC言語に類似。
- ・ただし、一般的なプログラム言語よりも完成度が低く、一部(だが重要な)の機能が無い。
(例: 関数、エラー・例外処理)
→ PyRAFの開発へ。
- ・PyRAFが未実装の環境や、引数を少しだけ変えてタスクを何度も繰り返し実行する場合などに、CLスクリプトはまだ利用機会がある。

自作タスク(=プログラム)の使用まで

自作タスクの使用までのステップ。

1. 作成: ファイル(**拡張子:cl**)に自作タスクのCLスクリプトを書く。
2. 登録: IRAFに自作タスクを登録する。
3. 使用: 登録した自作タスクを実行する。

作成(1): CLスクリプトの構造

procedure 自作タスク名(明示引数)

引数の定義(明示引数、隠し引数)

LDPの定義 # LDP(List-Directed Parameters, 後述)を使用する場合。

begin

変数の定義(型宣言) **#変数定義をここでまとめて行う!**

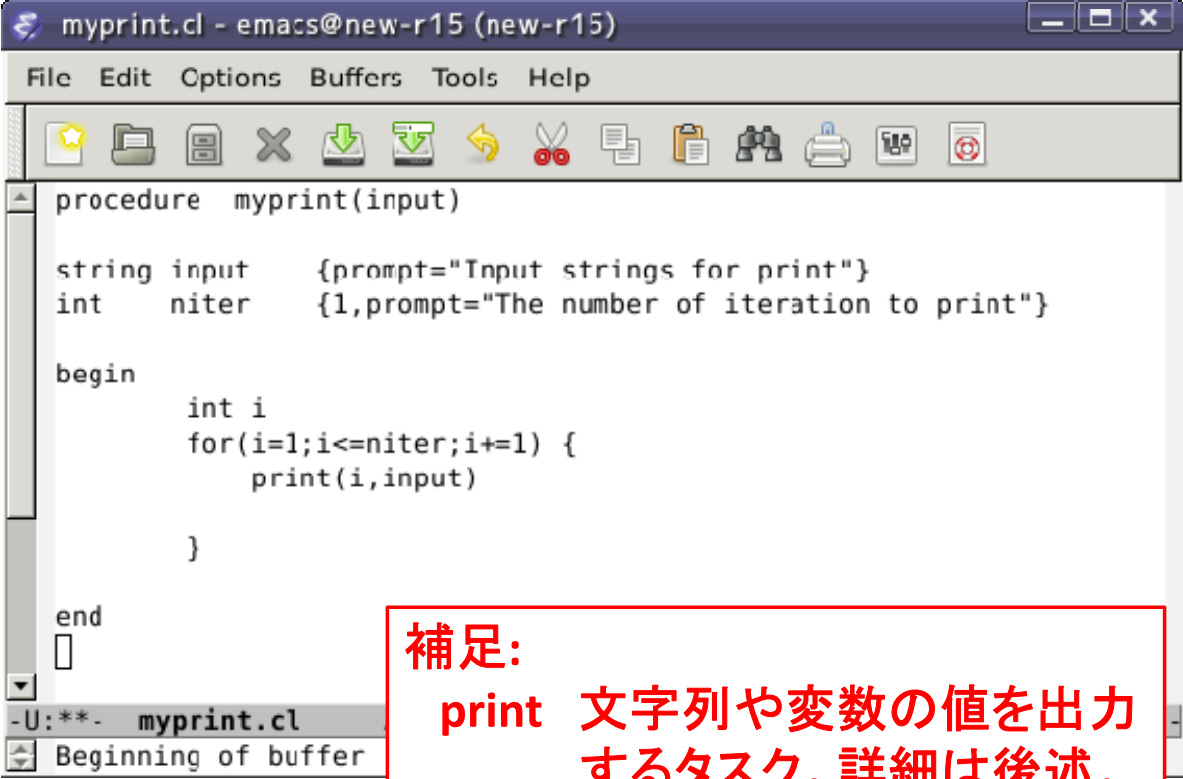
処理

end (必ず改行を入れること!)

補足: コメント文の挿入:「#」に続けて記述する。

作成(2): CLスクリプトの例

~/pylec/samples/clscripts/myprint.cl



```

procedure myprint(input)
  string input    {prompt="Input strings for print"}
  int    niter    {1,prompt="The number of iteration to print"}

  begin
    int i
    for(i=1;i<=niter;i+=1) {
      print(i,input)
    }
  end
end

```

補足:
print 文字列や変数の値を出力するタスク。詳細は後述。

procedure 自作タスク名(明示引数)

引数の定義

(明示引数、隠し引数)

begin

変数の定義(型宣言) #ここでまとめて!

処理

end #改行の挿入を忘れずに!

作成(3): タスクの記述方法(1)

IRAFでのタスクの記述方法は2種類ある。

コマンドモード: 対話的に実行する時に使用。

例: `imcombine @input.list output.fits comb=median`

コンピュータモード: CLスクリプト内でタスクを実行する時に使用。

例: `imcombine('@input.list', 'output.fits', comb='median')`

コンピュータモードは、引数の値に変数を使用可能。

例: `imstat(fn, for-, fi='midpt')` # 変数fnを入力ファイル名に設定。

作成(4): タスクの記述方法(2)

コンピュータモードの書式の詳細。

例: `imcombine('@input.list', 'output.fits', comb='median')`

コンピュータモードの書式:

- 文字列は「'」または「"」で囲む。 (変数の場合は囲まない)
- 引数の間に「,」を入れる。
- 引数全体を括弧「()」で囲む。

CLスクリプトでは、タスクをコンピュータモードで記述する。

作成(5): タスクの記述方法(3)

- ・ 1つのタスクは1行で記述する。
- ・ 1行を超えるようなタスクの記述には、行の末尾に「バックスラッシュ記号」の入力で、複数行にわたるタスクの記述が可能。

例:

```
imstat (inlist, fields="midpt", lower=INDEF, upper=INDEF, nclip=0, ¥  
lsigma=3., usigma=3.,binwidth=0.1, format=no, cache=no)
```

作成(6): タスクの記述方法(4)

タスクの引数は、**原則**全てを指定する。

利用者が過去に同じタスクを実行した際に指定した引数値の影響を排除するため。ユーザが変更することがない引数は除外しても良い。

(本テキストのサンプルスクリプトは可読性向上のため、必要最小限の指定)

例: displayタスク:

```
display (fn, 1, bpmask="BPM", bpdisplay="none", bpcolors="red", overlay="", ¥  
        ocolors="green", erase=yes, border_erase=no, select_frame=yes, repeat=no, ¥  
        fill=no, zscale=yes, contrast=0.25, zrange=yes, zmask="", nsample=1000, ¥  
        xcenter=0.5, ycenter=0.5, xsize=1., ysize=1., xmag=1., ymag=1., order=0, ¥  
        z1=INDEF, z2=INDEF, ztrans="linear", lutfile="")
```

作成(7): タスクの記述方法(5)

タスクの全引数指定の省力化の一例:

→ mkscriptタスク

```
vocl> mkscript
```

```
Script file name (mytask.cl): mytmp.cl
```

```
Task name of command to be added to script (imstat): display
```

→ epar display画面が表示されるので、適当に値を編集し、

「:wq」で抜ける。(※ z1, z2の値はINDEFを入力する)

→ 全引数を指定したタスクの記述が表示されるので、これを

コピー&ペーストして利用する。mkscriptタスクはCtrl-Cで強制終了する。

登録: 自作タスクの登録方法(1)

登録方法:

task (\$)タスク名 = ファイル名

タスクが引数を持たない場合、タスク名に「\$」をつける。

例: 引数無: task \$mytask = /home/studentXX/iraf/mytask.cl

引数有: task mytask = /home/studentXX/iraf/mytask.cl

一度登録したタスクを再登録する場合:

「task」の代わりに「**redefine**」を使う。

例: redefine (\$)mytask = /home/studentXX/iraf/mytask.cl

→ login.cl または loginuser.cl に記述すれば、IRAF起動時に登録済みになる。

登録: 自作タスクの登録方法(2)

例: loginuser.cl への登録:

(login.clではなく、loginuser.clでの登録を推奨)

登録手順:

1. ~/iraf/loginuser.clをテキストエディタで開く(新規作成)

2. 登録したいtaskを記述:

```
task ($)mytask = /home/studentXX/pylec/clscripts/mytask.cl
```

3. ファイルの末尾に「keep」を追記。改行を忘れずに!

例:

```
task mdisp1 = /home/studentXX/pylec/samples/clscripts/mdisp1.cl  
keep # 改行!
```

使用: 登録したタスクの実行方法

組み込みタスクの使用法と同じ。

別のCLスクリプト内で使用することも可能。

対話モードで使用する場合はコマンドモードで記述する。

CLスクリプトで使用する場合は前述のコンピュートモードで記述する。

先に覚えておくべきタスクたち

- **unlearn**

過去に実行したタスク変数のキャッシュをクリアする。

一度タスク登録をしたCLスクリプトを書き換えて、引数(明示・隠し問わず)を変更したら、必ず実行する。

例: `vocl> unlearn タスク名`

- **flprcache**

全ての実行中でないプロセスのキャッシュをクリアする。

例: `vocl> flprcache`

それでも動作がおかしいときは、IRAFをいったん終了し、再起動する。

データ型

文字列	string	(1023文字まで)
整数	int	(精度:32bit=10進数で9桁)
論理(bool)	bool	(yes/no または 1/0)
実数	real	(浮動小数点。倍精度)
struct型	struct	(scanやfscan関数などで使用する特殊な文字列型。後述)

変数の型宣言は、型名を使う:

例: 文字列変数「fnbody」の宣言

```
string fnbody
```

整数・実数型

四則演算:

加: $a + b$

減: $a - b$

乗: $a * b$

除: a / b

べき乗: $x ** y$

数学関数 $\sin(x)$, $\cos(x)$, $\exp(x)$, $\log(x)$, $\log_{10}(x)$, $\text{abs}(x)$ なども利用可能。

IRAFで利用可能な数学関数一覧は、「`vocl> help mathfcns`」を参照。

struct型の補足

struct型: scan関数やfscan関数などで使用する特殊な文字列型。

標準入力やファイルなどから文字列などを読み込むときに、通常の文字列型では空白を含めて受け取ることができない。

struct型の変数では、**空白を含めて受け取ることが可能**。

→ ファイルの一行(空白区切りで複数列のデータ)を、一つの変数に格納したい時などに有用。

試してみよう: (print、パイプ「|」、scanの詳細は後述)

```
vocl> string mystrg
```

```
vocl> print('a b c') | scan(mystrg)
```

```
vocl> print(mystrg)
```

→ a

```
vocl> struct mystct
```

```
vocl> print('a b c') | scan(mystct)
```

```
vocl> print(mystct)
```

→ a b c

型変換

以下の関数を使用することで、データ型の変換が可能。

文字列型への変換: `str(変数)`

int型への変換: `int(変数)`

real型への変換: `real(変数)`

引数

procedure 自作タスク名()の中に受け取る明示引数名を記載:

例: 明示引数 arg1, arg2 を持つタスク mytask の場合:

→ procedure mytask(arg1, arg2)

明示引数の型宣言はprocedure行の後、begin行の前に記載。

隠し引数もここに記載する。

例:

```
procedure mytask(arg1)
```

```
string arg1          {prompt="input argument1"} 明示引数
```

```
int    arg2          {5, prompt="option argument2"} 隠し引数
```

```
string hpar          {'default', prompt='hidden parameter 1'} 隠し引数
```

prompting

ユーザからの入力を促すプロンプトは、begin行の前の引数宣言の箇所に記述する。

例:

```
procedure mytask(arg1)
  string arg1      {prompt="input argument1"}  明示引数
  int    arg2      {5, prompt="option argument2"}  隠し引数
  string hpar      {'default', prompt='hidden parameter'}  隠し引数
```

隠し引数のデフォルト値は、{}の中、「prompt=」の前に記述する。

(例では、arg2の5やhparの'default'がデフォルト値)

プロンプトは、引数を使用するときに現れる。

List-Directed Parameters (LDP)

値としてファイルを指定すると、実行または呼び出し毎にファイルの中身を一行ずつ返す特殊な変数。

変数の前に「*」をつけて、文字列型で宣言する。 ← CLスクリプトでは「beginの前」で宣言!

例: `string *mylist1`

試してみよう:

```
vocl> string *mylist1 = 'home$login.cl'
```

```
vocl> =mylist1 (実行を繰り返す)
```

→ 実行毎に、~/iraf/login.clの中身が一行ずつ表示される。

CLスクリプトでは、**ファイルを読み込んで1行ずつ処理する際に使用。**

(実例は「ファイルからの読み込みで紹介」)

宣言済み変数

以下の変数は事前に宣言されているため、宣言なしで利用可能。
ただし、他の型の変数として宣言・使用できない(エラーとなる)。

例:

整数(int)型: i, j, k → 実は myprint.cl の「int i」は不要

実数(real)型: x, y, z

文字列(string)型: s1, s2, s3

論理(bool)型: b1, b2, b3

文字列型のLDP: list

struct型: line

変数一覧の確認: `vocl> epar cl`

値の出力・確認: print

文字列はもちろんのこと、変数の値やタスクの戻り値を表示できる。

タスクの動作確認、変数値のチェックなどスクリプトの作成時やデバッグ時に重宝。

用法: `print(...)` 「...」には'文字列'、変数、タスクなどを記述可能。

複数の文字列や変数を出力したい場合には、「,」でつなげる。

例:

`print(i)`

変数*i*の値を出力。

`print('# i= ', i)`

文字列「# i= 」と変数*i*の値を繋げて出力。

`print('!!! Error !!! file [', fn, '] is not found!')` 文字列、*fn*の値、文字列を出力。

`print(access('home$login.cl'))`

`access`(後述)の実行結果を出力。

補足: 類似タスクに`printf`(=書式指定`print`)あり。詳細は「`voctl> help print`」を参照。

標準入力からの読み込み: scan

標準入力から値を読み込み、変数に格納する。

用法: `scan(var1)` 標準入力から値を読み込み、変数`var1`に格納する。

`scan(var1, var2, ...)` 標準入力から値(空白区切りの)を読み込み、
変数`var1, var2, ...`に格納する。

例: `scan(s1)` 標準入力の値を読み込んで変数`s1`に格納する。

以下の類似タスクあり。詳細は「`vocl> help scan`」を参照。

fscan(`param, var1, var2, ...`) `param`から値を読み込み、変数に格納する。

→ ファイルからの読み込みで使用。

他、`scanf, fscanf` : 標準入力や変数から書式を指定して読み込む。

簡単な文字列の結合

簡単な文字列の結合: 結合演算子「//」

例:

- ・文字列同士の結合: 'ABC'//'def' → ABCdef
- ・文字列と数値の結合: 'S000'//595//'.fits' → S000595.fits
- ・文字列変数と数値変数の結合: fnbody//i//fnext → S000219.fits
(fnbody='S000', i=219, fnext='.fits'の場合)

より複雑な文字列操作: help str (substr, stridx, strldx, strlen, strlwr, struper,...)
を参照。

パイプ

パイプ「|」を使用可能。

パイプとscanタスクで、タスクの実行結果の変数への格納や、文字列の整形などを実現可能。

例:

- `printf('%05d¥n',i) | scan(s1)` → 数を整形する($i=50 \rightarrow 00050$)
- `print(s2) | sed("¥s/S/L/¥") | scan(s2)`

→ s2内の文字列から文字's'を文字'l'に置換。

(補足: sed:シェルコマンド、¥は「'」のエスケープで使用)

条件分岐: if 文

CLスクリプトのif文の書式:

```
if (条件式1) {  
    処理1  
}  
else if (条件式2) {  
    処理2  
}  
else {  
    処理3  
}
```

例:

```
if (i < 10) {  
    fn='S00'//i//'.fits'  
}  
else if (i < 100) {  
    fn='S0'//i//'.fits'  
}  
else {  
    fn='S'//i//'.fits'  
}
```

繰り返し: while文、for文

```
while (繰り返し条件式) {  
    処理  
}
```

```
for (初期化; 繰り返し条件; 変化式) {  
    処理  
}
```

```
例: while (i <= 100) {  
    fn='S000'//i//'.fits'  
    ...  
    i+=1  
}
```

```
例: for (i=1; i<imax; i+=1) {  
    fn='S000'//i//'.fits'  
    ...  
    print(fn)  
}
```

繰り返しの制御・スクリプトの終了

CLスクリプトにも、for文やwhile文の繰り返いを制御する命令が用意されている。

- ・ 次の繰り返しへの移動: next
- ・ 繰り返いの終了: break

条件文の中などでスクリプトを終了: bye

(PyRAFでは使えないので注意!)

ファイルからの読み込み

fscan関数と事前宣言済みのstring型LDP「**list**」、struct型変数「**line**」を使う。

例:

```
list = 'ファイル名'  
while( fscan(list,line) != EOF ) {  
    処理  
}
```

→ EOF = End of File: ファイル終端。

ファイルを1行ずつ読み込み、変数lineに格納し、処理を繰り返す。

ファイル終端まで読み込むと繰り返しを終了。

ファイルへの書込み

ファイルへの書込みは、リダイレクトを使用する:

- > file ファイル「file」への書込み
- >> file ファイルへの追記
- >& file ファイルへの通常およびエラー出力の書込み
- >>& file ファイルへの通常およびエラー出力の追記

例: imstatの出力結果(midpt)のファイル出力(ofn)

```
imstat(fn, for-, fi='midpt', > ofn)
```

論理演算子

if文などで条件を指定するときに、以下の演算子を使用可能:

論理積: &&

論理和: ||

論理否定: !

例:

if (式1 && 式2) → 式1と式2の両方を満たす場合

if (式1 || 式2) → 式1、式2のどちらかを満たす場合

if (!タスク) → bool型を返すタスクの真偽を反転させたい場合

(access, defpacなど。詳細は後述)

→ タスク == 1 or タスク != 1 (== 0) でも良い(真:1,偽:0)。

比較演算子

CLスクリプトの比較演算子は以下の通り:

aとbは一致 $a == b$ (数、文字列兼用)

aとbは不一致 $a != b$ (数、文字列兼用)

aはbより大きい $a > b$

aはbより小さい $a < b$

aはb以上 $a >= b$

aはb以下 $a <= b$

一時ファイル名の作成: mktemp

タスク内部でのみ使用するような、一時ファイル名の作成は `mktemp` タスクを使用すると便利。

用法: `tmpfn = mktemp('ファイル幹名')`

作成されるファイル名は、入力ファイル幹名+プロセス番号+1文字で、最後の1文字は呼び出しごとに変化。[例: `mktemp('tmp_')` → `tmp_9490e`]

`mktemp` タスクで作成した一時ファイルは、スクリプトを終了する前に削除する。

例: `delete(tmpfn)`

ワイルドカード(*,?,@)の使用: sections

IRAFのワイルドカード:

「*,?»などの普通のUNIXワイルドカードと、ファイルリストを展開する「@」

sectionsタスク:

IRAFワイルドカードを展開することが可能:

用法: sections(引数の変数, option='fullname', > 内部変数)

→ CLスクリプトに使用することで、柔軟な入力ファイルの指定が可能に。

自作タスクの使用では、sections + mktempを組み合わせると便利:

例: string tmlist=mktemp('temp_')
 sections(arg1, option='fullname', > tmlist)

ファイルのアクセス可否を判定: access

タスク実行前に**accessタスク**でファイルがアクセス可能(=読み込み可能)か確認し、不可であればタスクの実行を回避することで、スクリプトの異常終了を回避可能。

→ if文と組み合わせて使用。

用法: `access('ファイル名')` 返回值: bool型

```
例: if( access('ファイル名') ) {  
    タスク('ファイル名', ...)  
}
```

補足: 「不可(no)」を条件一致とする場合、「!」を `access` の前につける。

例: `if(! access(...))` (**bool型を返すタスクで共通**)

パッケージがload済みか判定: defpac

パッケージがload済みか判定する**defpac**タスクを使用することで、CLスクリプトで使用するタスクのパッケージを、loadされていない場合のみloadすることができる。これもif文と組み合わせて使う。

用法: `defpac('ファイル名')` 戻り値: bool型

```
例: if( ! defpac('パッケージ名') ) {  
    パッケージ名  
}
```

補足1: 「無い場合」を条件とする場合、`access`と同じく「!」をタスク名の前につける。

補足2: 類似タスク(`defpar`, `deftask`, `defvar`)あり。詳細は `vocl> help defpac` を参照。

2. CLスクリプト実習

実習データ

今回の実習で使用するデータ:

京都産業大学・神山天文台 1.3m 荒木望遠鏡 + 2色同時撮像装置ADLER
の「擬似」観測データ(人工的な処理を追加)

実習では1台のCCDカメラのデータ(2048 x 2048 pixels)を扱う。



京都産業大学・神山天文台



2色同時撮像装置ADLER

練習問題一覧 (CLスクリプト編)

CLスクリプト実習として、以下の練習問題を用意している:

1. 練習A1: 画像表示ソフトの作成
2. 練習A2: 画像表示ソフトの改良1
3. 練習A3: 不適切なデータ型の使用
4. * 練習A4: 画像表示ソフトの改良2
5. * 練習A5: 画像表示ソフトの改良3
6. * 練習A6: 画像表示ソフトの改良4
7. 練習A7: 画像表示ソフトの改良5

補足: 「*」付きの問題(A4, A5, A6)は「**時間に余裕がある人**」向け。

これらの問題を飛ばしても最後の問題(練習A7)に取り組める。

練習A1: 画像表示ソフトの作成

フレーム名のリストを受け取り、3秒間隔でds9にフレームを表示するタスクを作る。
作成したら登録・実行する(以降の練習も同様)。

タスク名: mdisp1

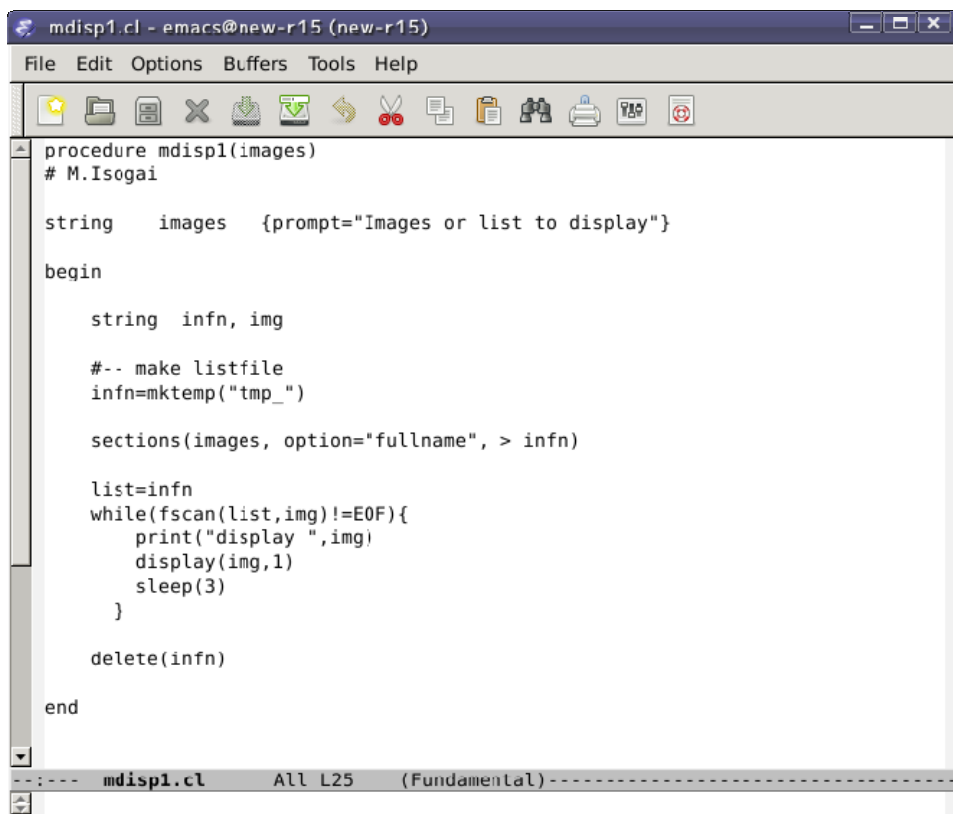
明示引数の名前: images

使用タスク: mktemp, sections, display, sleep

補足: sleep: IRAFの組み込み関数。用法: sleep(秒数)

動作の詳細: 一時ファイル名を作成。引数をsectionsで展開して一時ファイルに格納。
このファイルを一行ずつ読み込んで繰り返し処理(フレーム名の表示、
画像の表示、sleep)をする。実行前にIRAFに登録し、ds9を起動しておく。

練習A1のスクリプト例:



```
mdisp1.cl - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Help
procedure mdisp1(images)
# M. Isogai
string images {prompt="Images or list to display"}
begin
  string infn, img
  #-- make listfile
  infn=mktemp("tmp_")
  sections(images, option="fullname", > infn)
  list=infn
  while(fscan(list, img)!=EOF){
    print("display ", img)
    display(img, 1)
    sleep(3)
  }
  delete(infn)
end
```

~/pylec/samples/clscripts/mdisp1.cl

登録例:

```
task mdisp1 = /home/studentXX/  
pylec/samples/clscripts/mdisp1.cl
```

(実際には一行に記述)

実行例:

```
vocl> cd /home/studentXX/pylec/data/100918  
vocl> !ds9&  
vocl> mdisp1 @object.list
```

練習A2: 画像表示ソフトの改良1

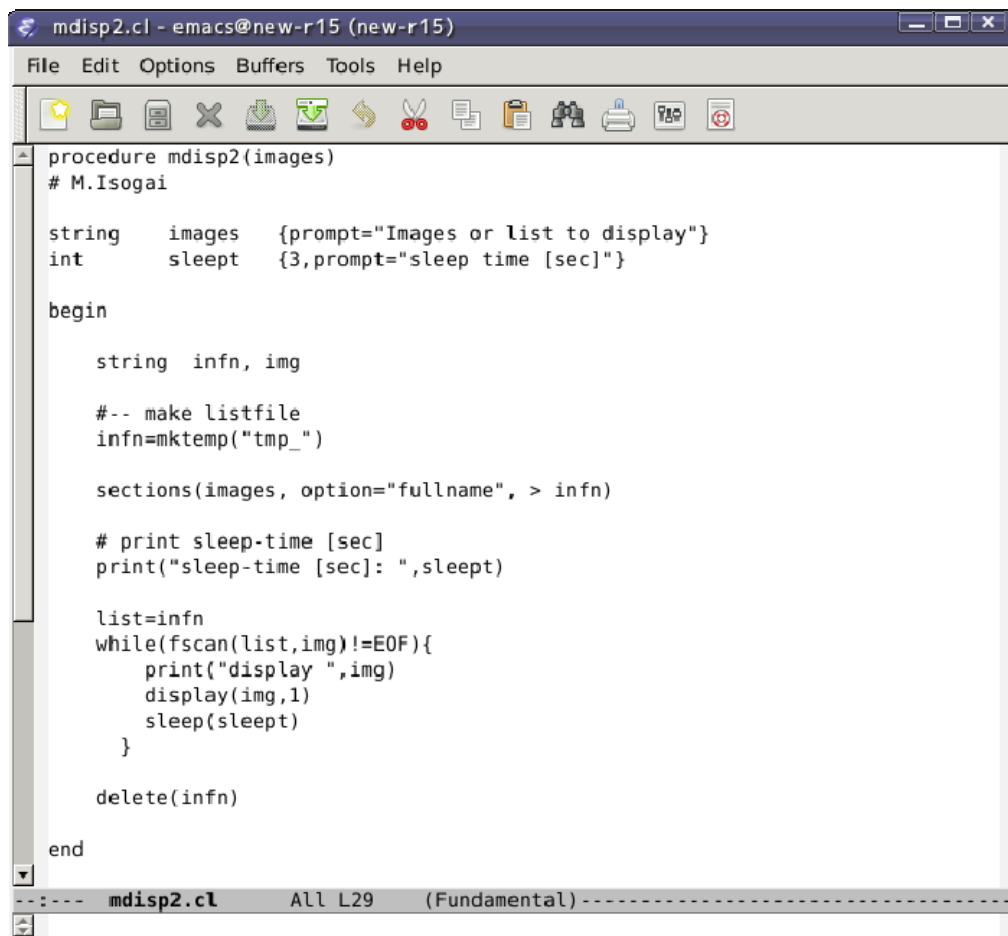
練習A1で作成した画像表示ソフトを改良して、表示間隔を隠し引数 (sleep, int型, デフォルト値は3)として追加し、表示間隔の設定値を表示する。

(mdisp2.cl)

実行例: `mdisp2 @object.list sleep=5`

(※ タスクを登録してから実行する。これ以降の練習も同様)

練習A2のスクリプト例:



```
procedure mdisp2(images)
# M.Isogai

string  images  {prompt="Images or list to display"}
int     sleep   {3,prompt="sleep time [sec]"}

begin

  string  infn, img

  #-- make listfile
  infn=mktemp("tmp_")

  sections(images, option="fullname", > infn)

  # print sleep-time [sec]
  print("sleep-time [sec]: ",sleep)

  list=infn
  while(fscan(list,img)!=EOF){
    print("display ",img)
    display(img,1)
    sleep(sleep)
  }

  delete(infn)

end
```

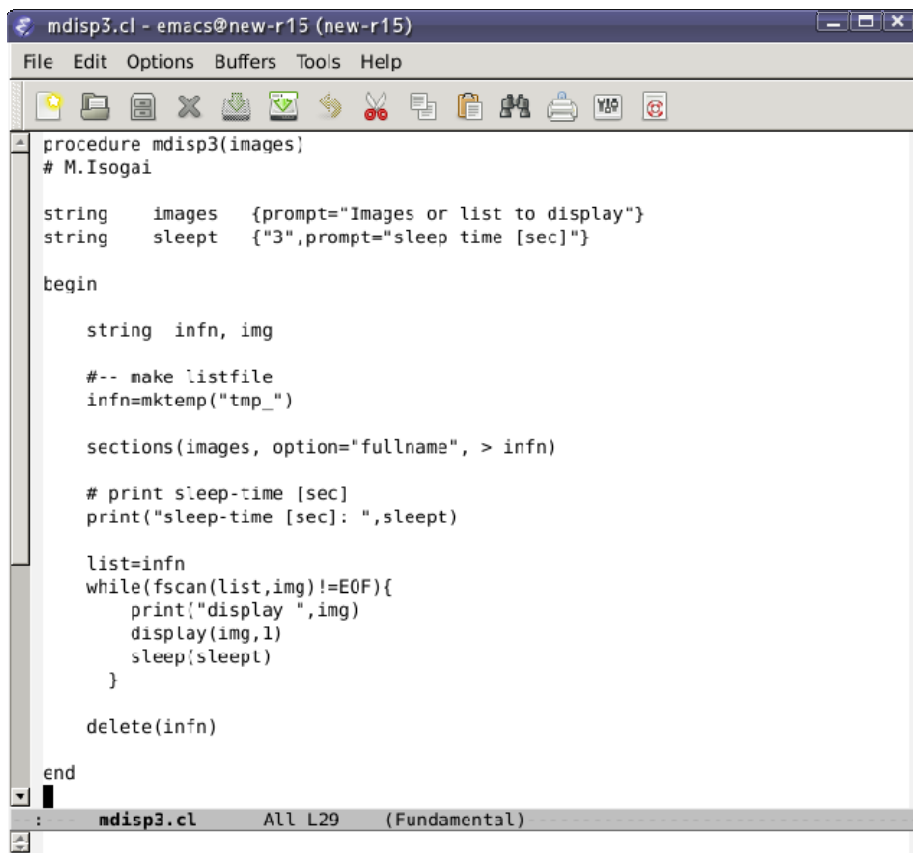
~/pylec/samples/clscripts/mdisp2.c1

練習A3: 不適切なデータ型の使用

練習A2で追加した隠し引数のデータ型をint型からstring型に変更し、動作を確認する(`mdisp3.cl`)。

実行例: `mdisp3 @object.list slept=5`

練習A3のスクリプト例:



```
mdisp3.cl - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Help
procedure mdisp3(images)
# M.Isogai

string images {prompt="Images or list to display"}
string sleep  {"3",prompt="sleep time [sec]"}

begin

  string infn, img

  #-- make listfile
  infn=mktemp("tmp_")

  sections(images, option="fullname", > infn)

  # print sleep-time [sec]
  print("sleep-time [sec]: ",sleep)

  list=infn
  while(fscan(list,img)!=EOF){
    print("display ",img)
    display(img,1)
    sleep:sleep)
  }

  delete(infn)

end
: mdisp3.cl All L29 (Fundamental)
```

~/pylec/samples/clscripts/mdisp3.cl

実行結果:

1枚目の画像が表示された後、「48+sleep」秒経過した後に2枚目の画像が表示される(はず)。

CLスクリプトでタスクの引数として使う変数を使用する際には、**タスクが許容する型と同じ型で宣言する**必要がある。

(もしくは、タスクが許容する型に変換してからタスクで使用する)

練習A4: 画像表示ソフトの改良2

画像表示ソフトの隠し引数として、displayタスクの表示に関する4つの隠し引数 (zscale=yes, zrange=yes, z1=1000, z2=2000)を(引数名を)大文字にして追加する (mdisp4)。

[補足: z1, z2: 画像表示で黒・白(グレースケールの場合)とするカウント値。

自動割当(zscale, zrange)をnoに設定しないと機能しない。

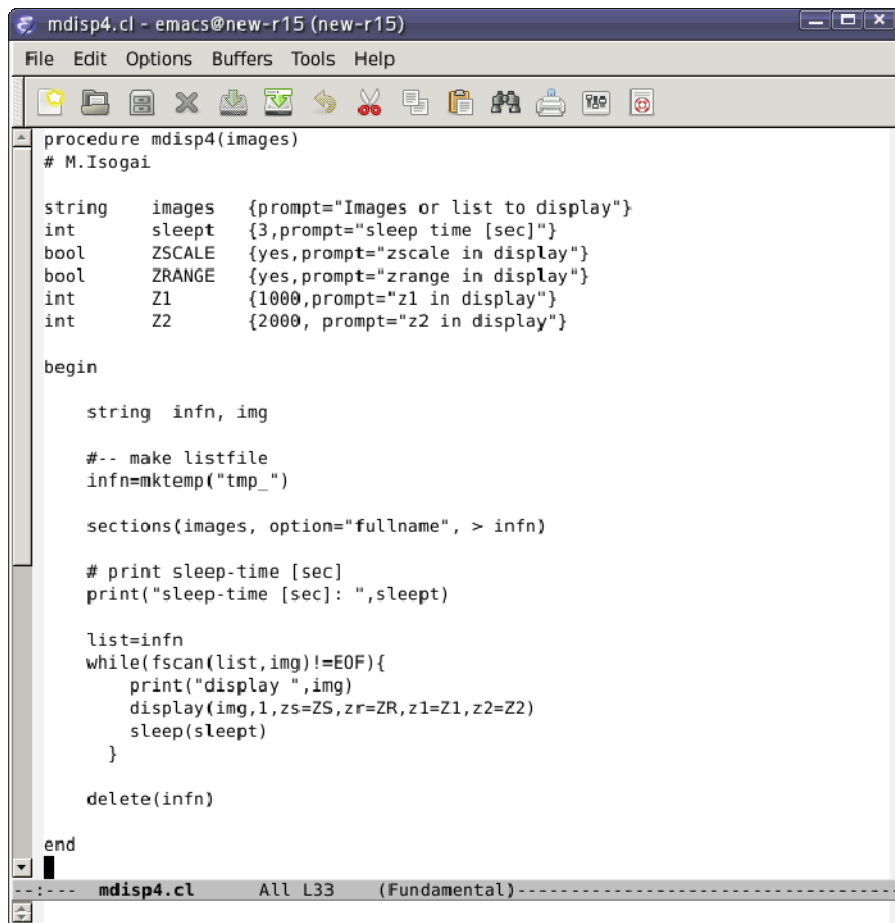
詳細は `vocl> help display` を参照。]

隠し引数名を大文字に変えるのは、displayの隠し引数との混同を避けるため。

実行例: `mdisp4 @object.list ZS- ZR- Z1=1000 Z2=2000`

※ 余裕がある受講生向け

練習A4のスク립ト例:



```
procedure mdisp4(images)
# M.Isogai

string  images  {prompt="Images or list to display"}
int     slept   {3,prompt="sleep time [sec]"}
bool    ZSCALE  {yes,prompt="zscale in display"}
bool    ZRANGE  {yes,prompt="zrange in display"}
int     Z1      {1000,prompt="z1 in display"}
int     Z2      {2000, prompt="z2 in display"}

begin

  string infn, img

  #-- make listfile
  infn=mktemp("tmp_")

  sections(images, option="fullname", > infn)

  # print sleep-time [sec]
  print("sleep-time [sec]: ",slept)

  list=infn
  while(fscan(list,img)!=EOF){
    print("display ",img)
    display(img,1,zs=ZS,zr=ZR,z1=Z1,z2=Z2)
    sleep(slept)
  }

  delete(infn)

end
```

~/pylec/samples/clscripts/mdisp4.c

練習A5: 画像表示ソフトの改良3

画像表示ソフトに、画像の統計情報の出力を追加する(mdisp5)。
(挿入場所: 画像を表示した後、sleepの前)

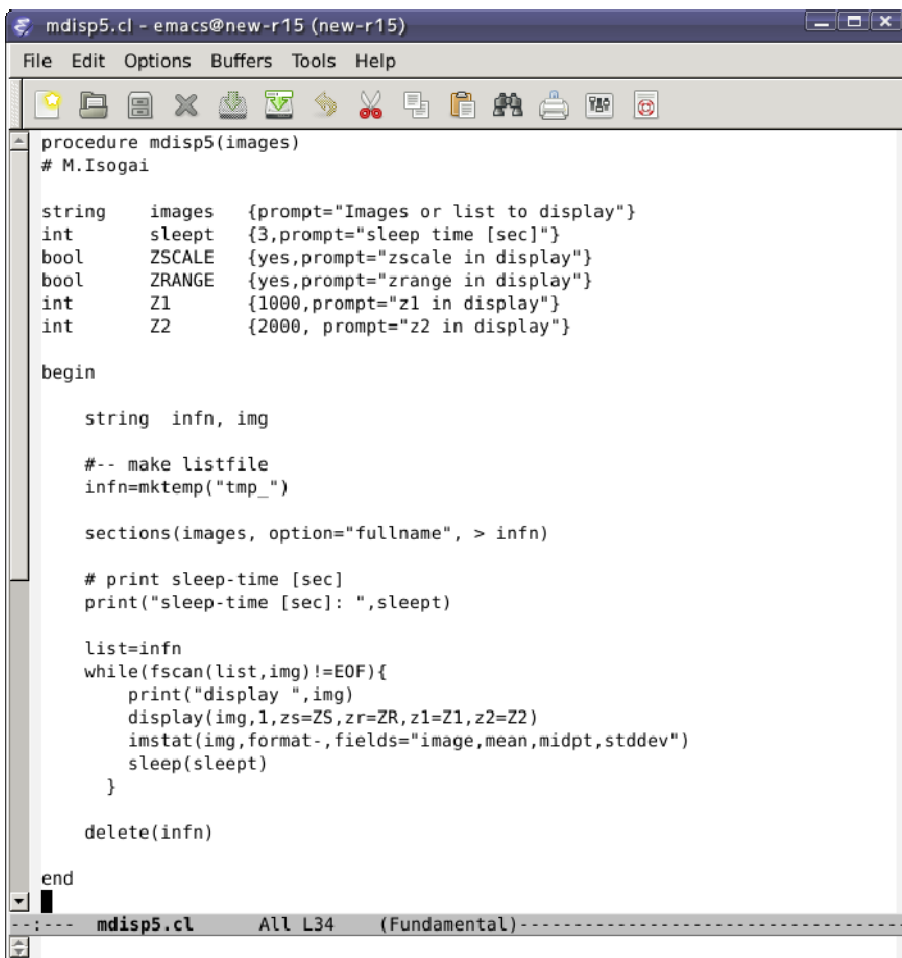
使用タスク: imstat

出力: 画像名ファイル名、平均値、中央値、標準偏差

実行例: `mdisp5 @object.list`

※ 余裕がある受講生向け

練習A5のスク립ト例:



```
procedure mdisp5(images)
# M.Isogai

string  images  {prompt="Images or list to display"}
int     sleept   {3,prompt="sleep time [sec]"}
bool    ZSCALE  {yes,prompt="zscale in display"}
bool    ZRANGE  {yes,prompt="zrange in display"}
int     Z1      {1000,prompt="z1 in display"}
int     Z2      {2000, prompt="z2 in display"}

begin

  string infn, img

  #-- make listfile
  infn=mktemp("tmp_")

  sections(images, option="fullname", > infn)

  # print sleep-time [sec]
  print("sleep-time [sec]: ",sleept)

  list=infn
  while(fscan(list,img)!=EOF){
    print("display ",img)
    display(img,1,zs=ZS,zr=ZR,z1=Z1,z2=Z2)
    imstat(img,format-, fields="image,mean,midpt,stderr")
    sleep(sleept)
  }

  delete(infn)

end
```

~/pylec/samples/clscripts/mdisp5.cl

練習A6: 画像表示ソフトの改良4

CLスクリプト内で使用するパッケージ「tv」がloadされているか確認し、loadされていればメッセージを、なければloadする機能を追加する (mdisp6)。

使用するタスク: `defpac('パッケージ名')`

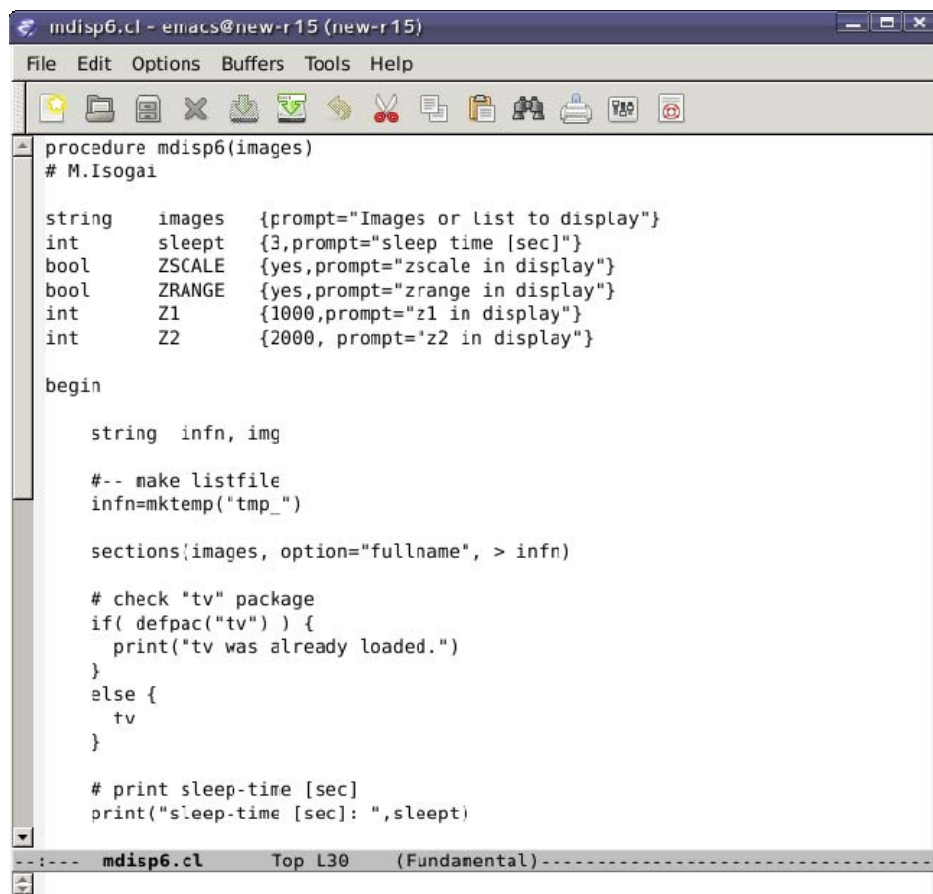
指定したパッケージがload済みか判定。返り値: bool型。

実行例: `mdisp6 @object.list`

※ 余裕がある受講生向け

練習A6のスク립ト例:

~/pylec/samples/clscripts/mdisp6.cl



```
mdisp6.cl - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Help
procedure mdisp6(images)
# M.Isogai

string images {prompt="Images or list to display"}
int sleep {3,prompt="sleep time [sec]"}
bool ZSCALE {yes,prompt="zscale in display"}
bool ZRANGE {yes,prompt="zrange in display"}
int Z1 {1000,prompt="z1 in display"}
int Z2 {2000, prompt="z2 in display"}

begin

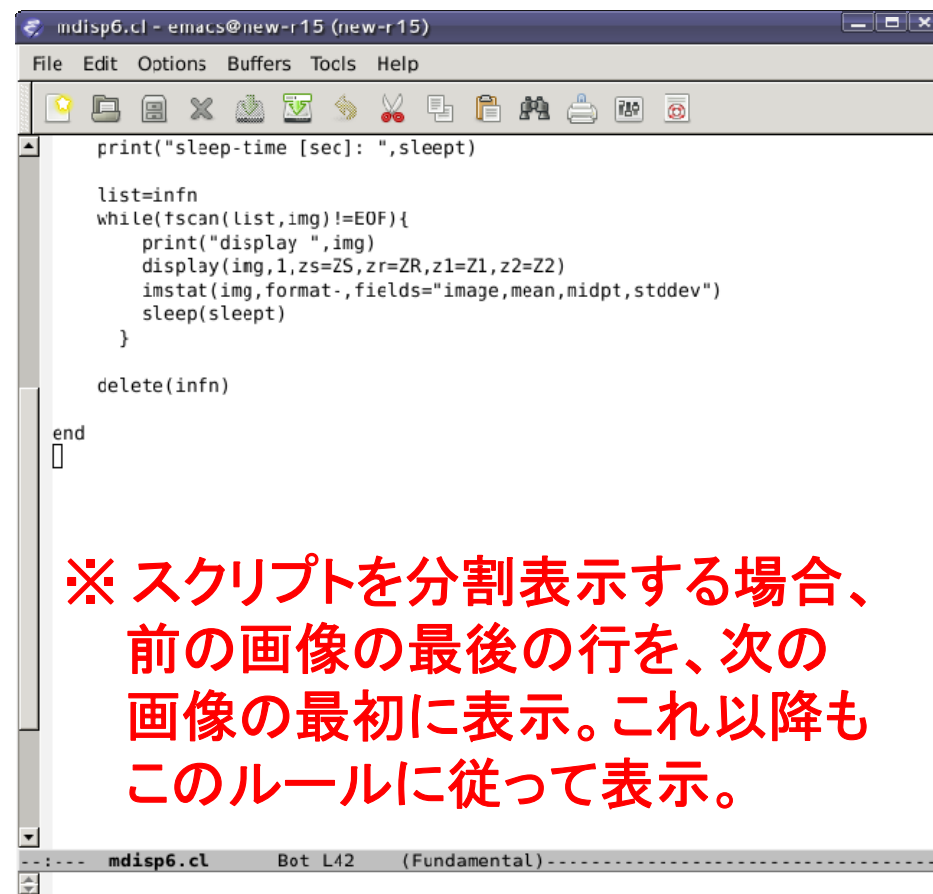
string infn, img

#-- make listfile
infn=mktemp("tmp_")

sections{images, option="fullname", > infn)

# check "tv" package
if( defpac("tv") ) {
print("tv was already loaded.")
}
else {
tv
}

# print sleep-time [sec]
print("sleep-time [sec]: ",sleep)
```



```
mdisp6.cl - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Help

print("sleep-time [sec]: ",sleep)

list=infn
while(fscan(list,img)!=EOF){
print("display ",img)
display(img,1,zs=ZS,zr=ZR,z1=Z1,z2=Z2)
imstat(img,format-,fields="image,mean,midpt,stddev")
sleep(sleep)
}

delete(infn)

end
```

※ スクリプトを分割表示する場合、
前の画像の最後の行を、次の
画像の最初に表示。これ以降も
このルールに従って表示。

練習A7: 画像表示ソフトの改良5

ソフトで表示する画像ファイルが存在するか画像を表示する前に確認し、なければ警告文を表示し、その画像の表示をスキップする機能を追加する。
(練習A2の `mdisp2.cl` をベースに)

使用するタスク: `access('ファイル名')`

指定したファイルがアクセス可能(=読み込み可能)か判定。

返り値: `bool`型。

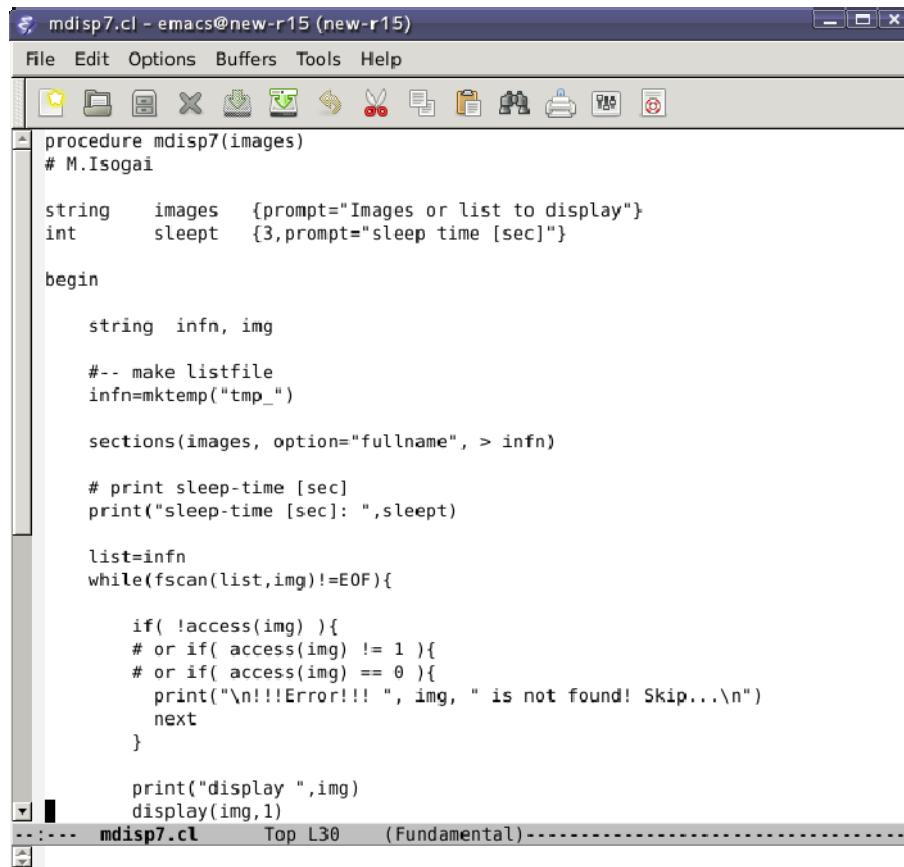
補足: 否定の条件では「!」を使うか、「!= 1」または「== 0」を使う。

(詳細は論理演算子と`access`の項を参照)

実行例(100918): `mdisp7 @object.dummy.list`

練習A7のスク립ト例:

~/pylec/samples/clscripts/mdisp7.cl



```
mdisp7.cl - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Help
procedure mdisp7(images)
# M.Isogai

string  images {prompt="Images or list to display"}
int     sleept  {3,prompt="sleep time [sec]"}

begin

  string infn, img

  #-- make listfile
  infn=mktemp("tmp_")

  sections(images, option="fullname", > infn)

  # print sleep-time [sec]
  print("sleep-time [sec]: ",sleept)

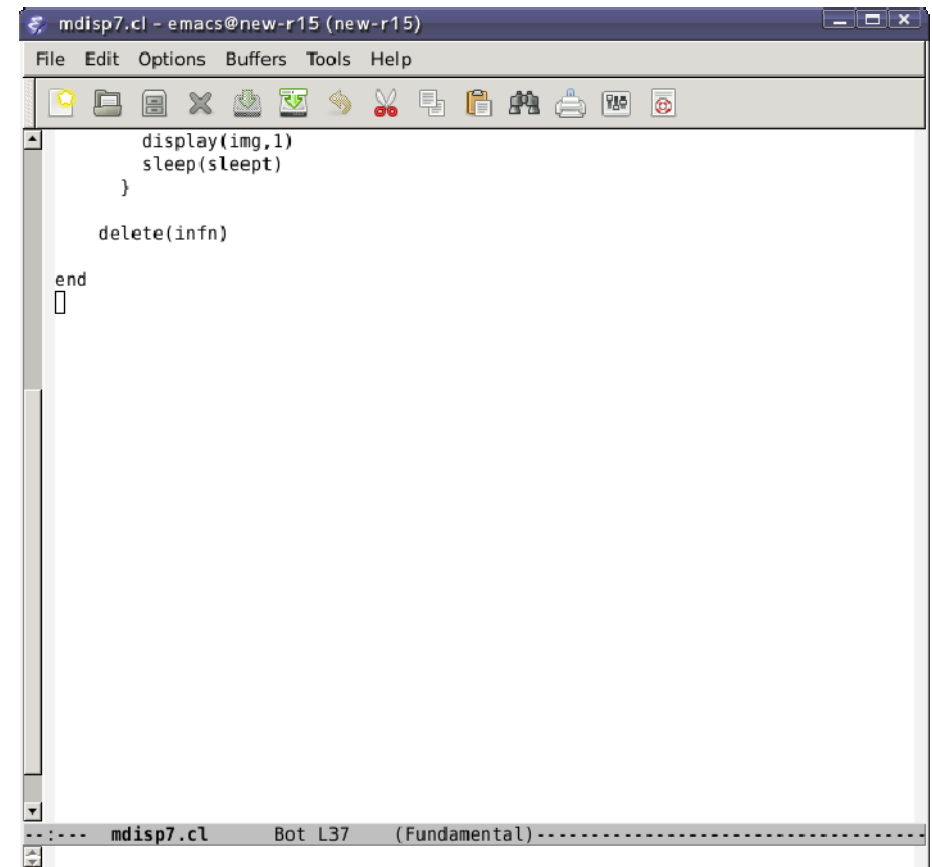
  list=infn
  while(fscan(list,img)!=EOF){

    if( !access(img) ){
      # or if( access(img) != 1 ){
      # or if( access(img) == 0 ){
      print("\n!!!Error!!! ", img, " is not found! Skip...\n")
      next
    }

    print("display ",img)
    display(img,1)
  }
}

```

mdisp7.cl Top L30 (Fundamental)



```
mdisp7.cl - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Help
    display(img,1)
    sleep(sleept)
  }

  delete(infn)

end

```

mdisp7.cl Bot L37 (Fundamental)

CLスクリプト編の参考文献(1)

- An Introductory User's Guide to IRAF Scripts

<ftp://iraf.noao.edu/ftp/docs/script.pdf>

古い(1989年, v2.8用)が今でも有効。

- IRAF CL Script Tips & Tricks

ftp://iraf.noao.edu/ftp/docs/script_intro.pdf

比較的新しい(2003年)。プレゼンファイル。例が豊富。

- 過去のADC IRAF講習会テキスト:

http://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo.html#iraf_prog

このテキスト作成の参考資料でもある。

- IRAFのhelp (help language, intro, str, paraなど)

CLスクリプト編の参考文献(2)

- ・ IRAFノウハウ集(FAQ) (埼玉大 大朝氏のWebページ)

http://www.astron.sci.edu.saitama-u.ac.jp/iraf_knowhow.html

天文情報処理研究会にあったページの復活版(?)

- ・ IRAF TIPS (広島大 秋田谷氏のWebページ)

<http://home.hiroshima-u.ac.jp/akitaya/research/memo/iraftips.html>

以上でCLスクリプト実習は
終了です。

次からは、いよいよPyRAFを
学んでいきます。

3. PyRAFについて

PyRAFとは？

- ・ IRAFのタスクをPythonスクリプトで使用できるようにしたソフトウェア。
- ・ STScIが開発: http://www.stsci.edu/institute/software_hardware/pyraf
- ・ 対話的に実行できるCLエミュレーションモード(以下対話モード)とPythonスクリプト内でモジュールとして利用するPythonモードの両方の機能を備える。
- ・ 対話モードは、IRAFのCLとほぼ同じように利用可能。
- ・ PyRAFの利用には、IRAFがインストールされている必要あり。

IRAF CLとの違い(1)

- ・ PyRAFは独自のグラフ描写カーネルを持つ。
→ xgterm不要。
- ・ IRAFのタスク名と同名のコマンドがPythonにある場合、pythonのコマンドが優先される。頻繁に起きるのはprintとdelete:
IRAFのprint, deleteを使用する方法:
print -> **cl**Print delete -> delete or dele
- ・ IRAFのタスク名がPythonの予約語と一致する場合、「IRAFのタスクを使用する」には、タスク名の前に**PY**をつけて実行する。
例: irafのimportの実行: **PY**import

IRAF CLとの違い(2)

- help表示:

help タスク名など -> IRAFのヘルプが表示される。

Pythonのヘルプ: help()

- PyRAFではパッケージをアンロードできない:

bye, keepコマンドは存在するが、実際には何もしない。

- CLスクリプトのGOTO宣言がない。

Pythonにはgoto宣言がないので、goto宣言を使用した自作CLスクリプトは

PyRAFでは正常に動作しない。

IRAF CLとの違い(3)

- ・ バックグラウンドでの実行ができない。
CLスクリプトのバックグラウンド実行は無視される。
- ・ CLスクリプトのエラートレースバックの行番号
CLスクリプトを実行した際のエラートレースバックに表示される行番号はPythonに変換されたスクリプトの行番号で、元のCLスクリプトの行番号ではない。

変換スクリプトの表示: `print iraf.自作タスク名.getCode()` など。

例(mdisp1.cl): `print iraf.mdisp1.getCode()`

(※行番号は表示されない)

PyRAFの使い方(1)

まずは対話モードで使ってみよう。

ターミナルを立ち上げ、

```
$ pyraf [enter]
```

で起動。

(IRAF voclと同様、補完・履歴機能が完備)

・PyRAFの終了:「.exit」

例: pyrafの起動: (次ページへ)

\$ pyraf

setting terminal type to 'xgterm' ...

NOAO/IRAF PC-IRAF Revision 2.16.1 EXPORT Mon Oct 14 21:40:13 MST 2013
This is the EXPORT version of IRAF V2.16 supporting PC systems.

Welcome to IRAF. To list the available commands, type ? or ??. To get detailed information about a command, type `help <command>'. To run a command or load a package, type its name. Type `bye' to exit a package, or `logout' to get out of the CL. Type `news' to find out what is new in the version of the system you are using.

Visit <http://iraf.net> if you have questions or to report problems.

*** Checking update status... Your IRAF system is up to date
*** Using global login file: /home/studentXX/iraf/login.cl
The following commands or packages are currently defined:

clpackage/:

clpackage/ language/ plot/ system/ vo/
dataio/ lists/ proto/ tables/
dbms/ noao/ softools/ user/
images/ obsolete/ stsdas/ utilities/

PyRAF 2.1.10 Copyright (c) 2002 AURA

Python 2.7.11 Copyright (c) 2001-2015 Python Software Foundation.

Python/CL command line wrapper

.help describes executive commands

--> **# プロンプトが出る。**

PyRAFの使い方(2)

PyRAF起動時には、IRAF起動時と同じメッセージの後、PyRAFの起動メッセージが出力される。

これは、起動時にIRAFの環境設定ファイルを参照するため。

参照の順番: 1: ./login.cl, 2: ~/iraf/login.cl

IRAFホームディレクトリが~/irafの場合、どのディレクトリからでもPyRAFを起動可能。
(実習の最初でIRAFのホームディレクトリを~/irafとしたのは、このため)

試してみよう(1)

以下を実行してみよう:

--> **!ds9&**

--> **displ dev\$pix 1** (→ ds9にM51の画像が表示される)

--> **imhe dev\$pix**

→ dev\$pix[512,512][short]: m51 B 600s

--> **imhe dev\$pix l+**

→ (全ヘッダが表示される)

--> **imstat dev\$pix**

→ #	IMAGE	NPIX	MEAN	STDDEV	MIN	MAX
	dev\$pix	262144	108.3	131.3	-1.	19936.

試してみよう(2)

--> `imstat dev$pix for-`

→ `dev$pix 262144 108.3154 131.298 -1. 19936.`

--> `imstat dev$pix for- fi=mid`

→ `88.74712`

--> `imstat[tab] dev$pix for[tab]- fi[tab]=mid` # [tab] = tabキーを押す

→ 「tabキー」でタスク名や変数名の補完が可能。

(上の例では、`imstatistics dev$pix format- fields=mid` となるはず)

EPAR 変数エディタ(1)

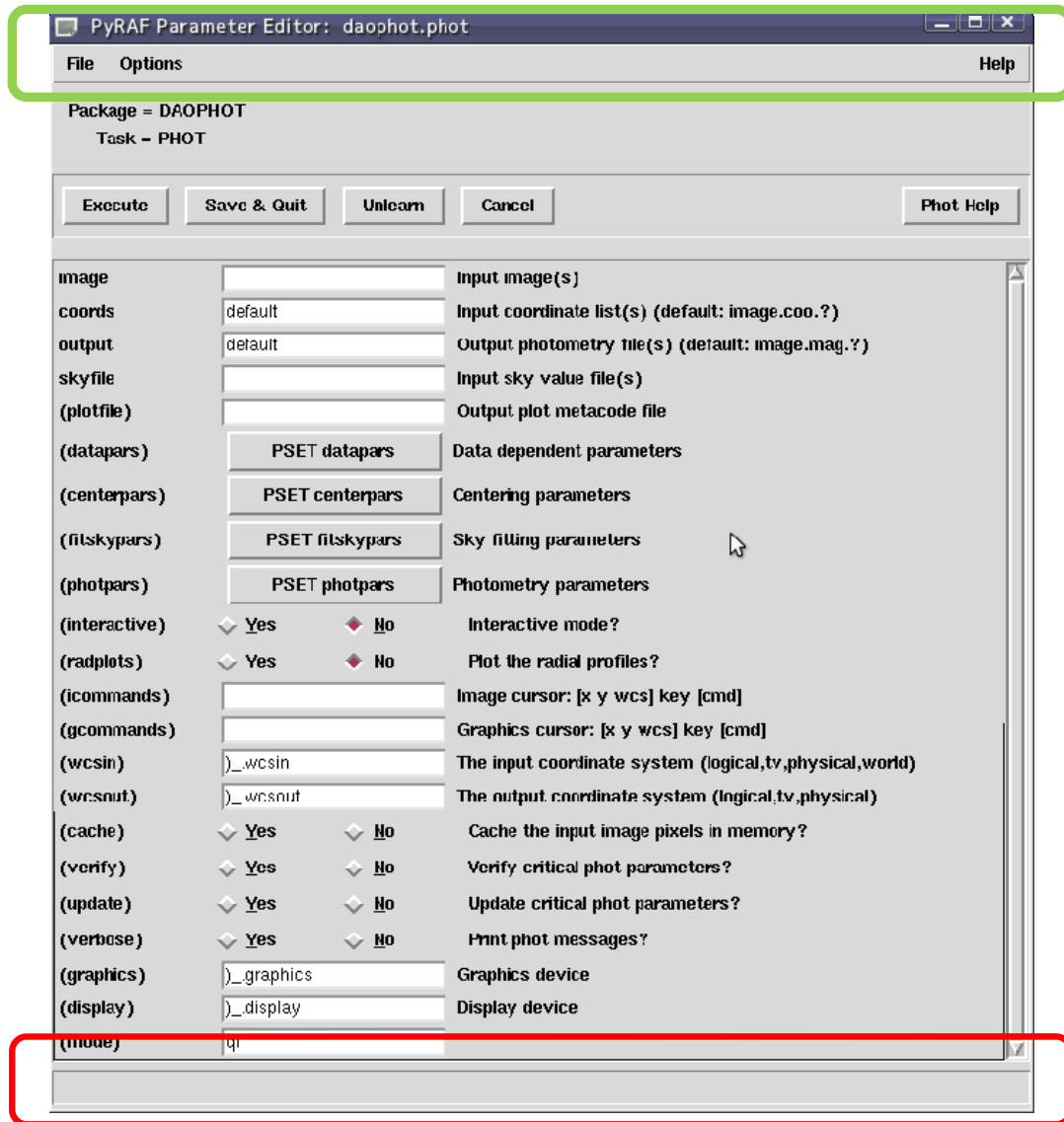
- epar タスク名で変数編集GUIが起動する。

例: daophot.phot

```
--> daophot      # daophotパッケージのload
```

```
daophot/:
```

```
addstar      daotest      nstar        pexamine     psf
allstar      datapars@    pcalc        pfmerge      psort
centerpars@  findpars@    pconcat      phot         pstselect
daoedit      fitskypars@  pconvert     photpars@    seepsf
daofind      group        pdump        prenumber    setimpars
daopars@     grpselect    peak         pselect      substar
--> epar phot
```



メニューバー:

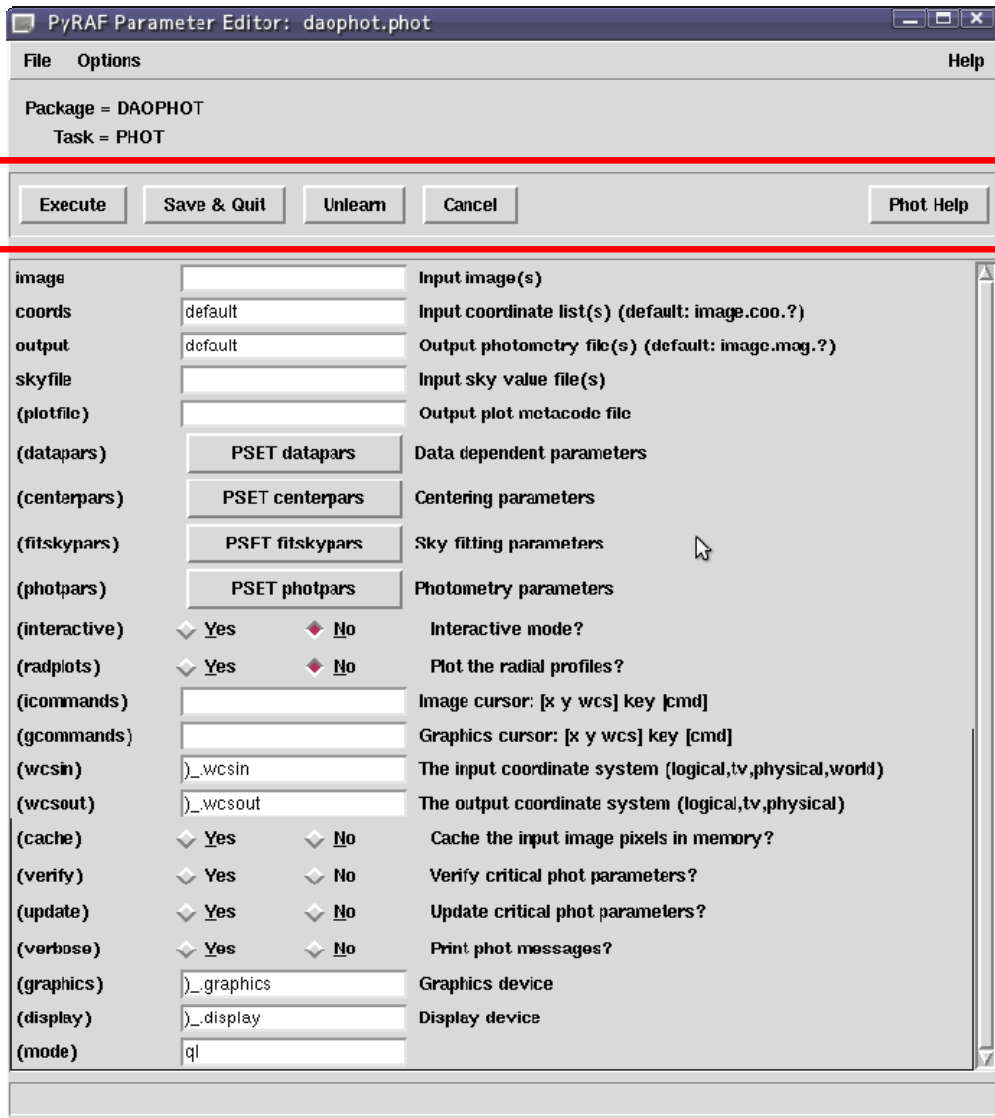
File: アクションボタン機能を直接選択可能

Options: ヘルプの表示方法を選択可能:
(ウインドウ or ブラウザ)

Help: このタスクのヘルプかEPARのヘルプを
表示可能

ステータスライン:

アクションボタンのヘルプ情報や、入力値の
チェック結果(最後の)が表示される。



アクションボタン:

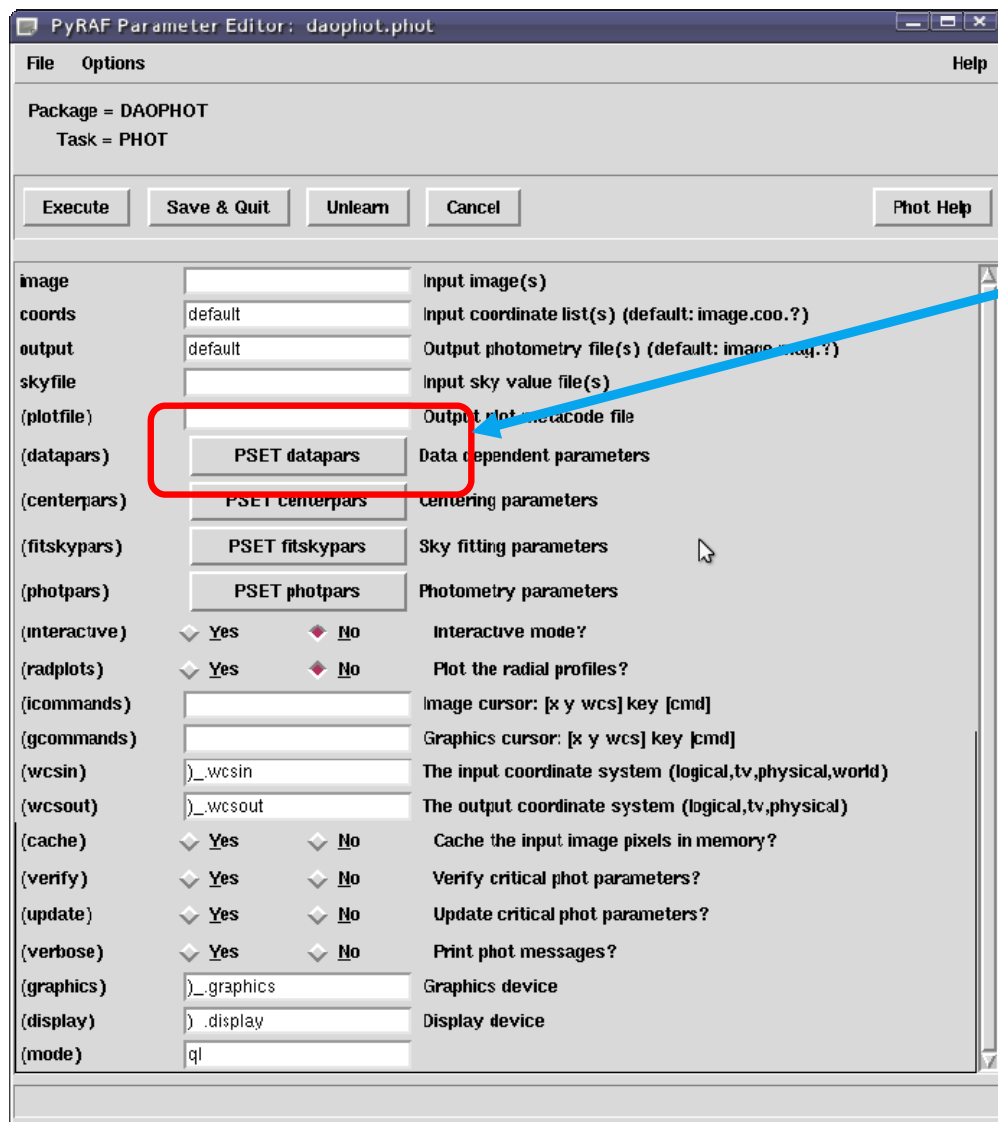
Execute: 現在表示されている変数値で
タスクを即座に実行。

Save&Quit: 現在の変数値を保存して終了。

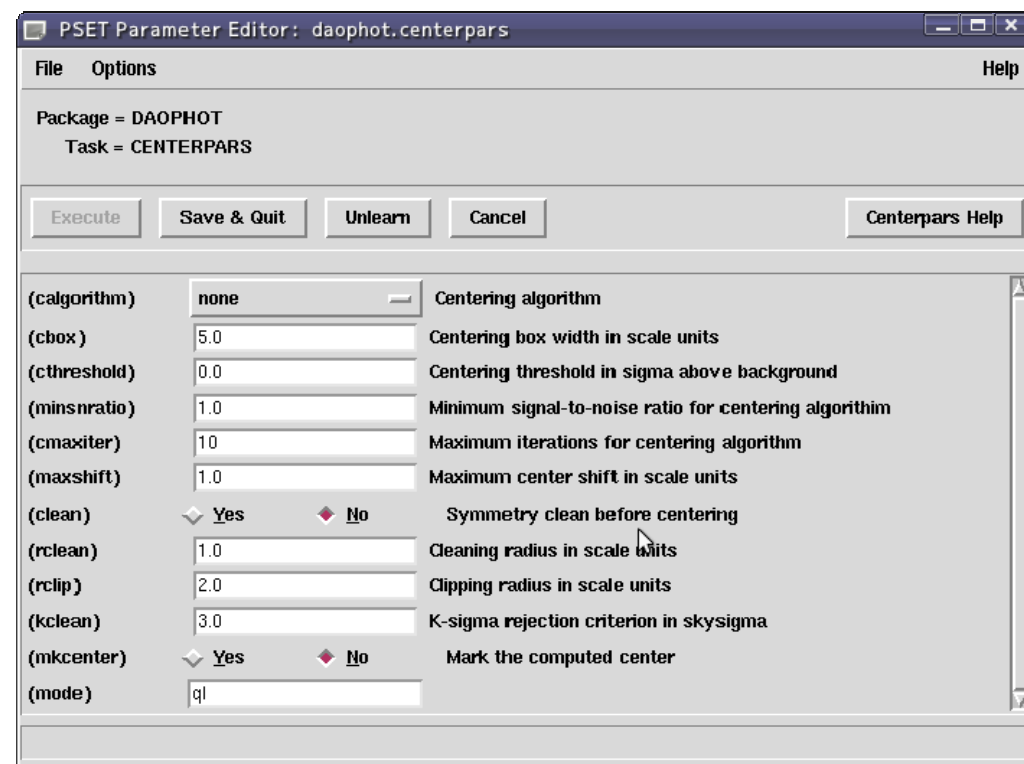
Unlearn: すべての変数値をシステムの
デフォルト値に戻す。

Cancel: 現在の変数値を保存せずに終了。

「タスク名 help」: タスクのhelpを別ウインドウで表示。



PSET ボタン:
 クリックするとPSETウィンドウが開く。
 親ウィンドウと同時に編集可能。



EPAR変数エディタ(2)

入力情報: 別の入力ボックスがアクティブなときにチェックされる。

入力ボックスでマウスの右クリック:

→ ポップアップメニューが開く:

ファイルブラウザ

入力ボックスのクリア

入力ボックスの変数のみのunlearn

PyRAFのグラフ描写

- ・PyRAFは独自のグラフ描写カーネルを持つため、xgtermを使う必要がない。

(一部の機能はIRAFのグラフィックカーネルを使用。例: 印刷)

- ・IRAFと完全に同一ではないが、多くの同等機能を持つ。

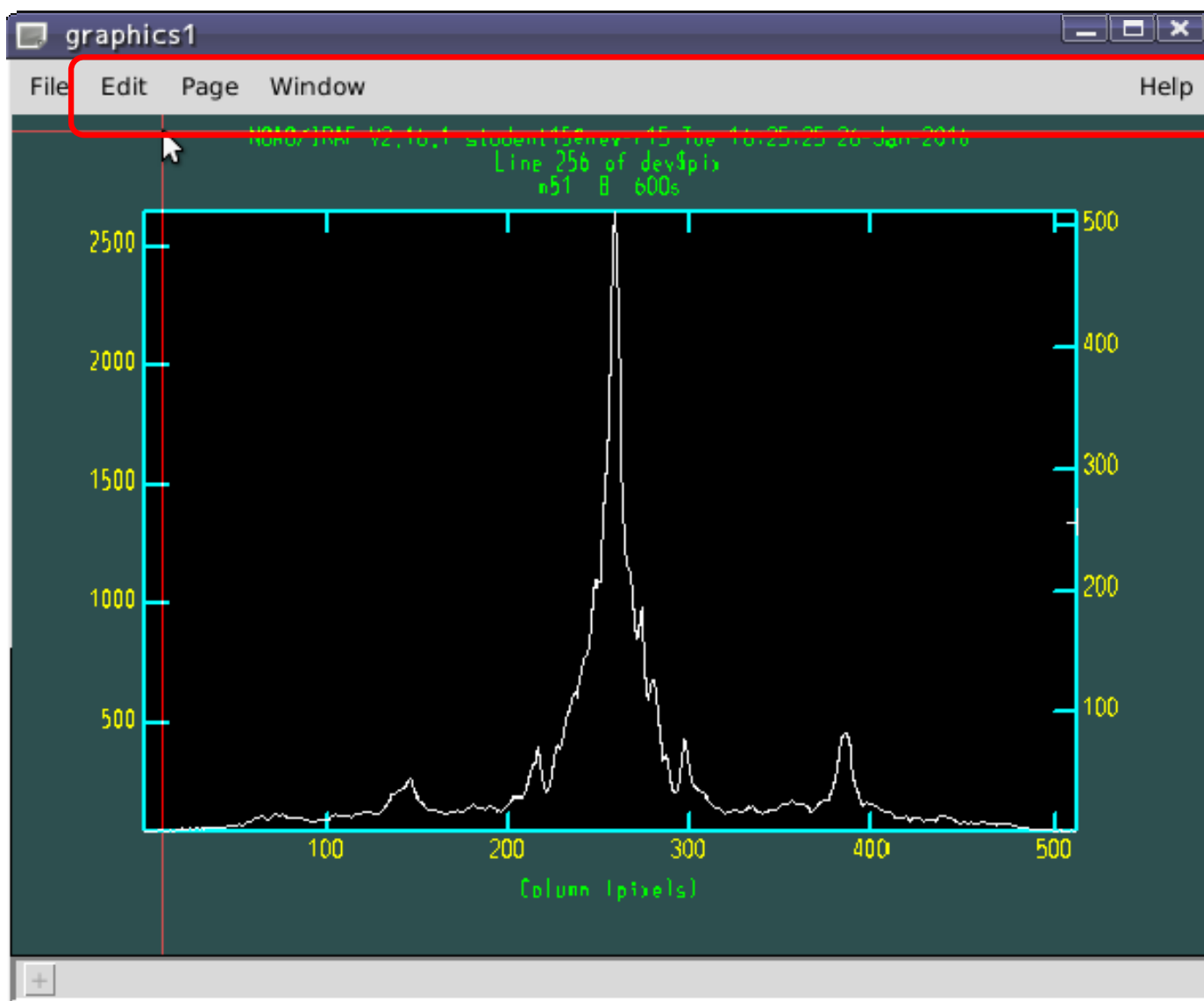
- ・IRAFと違い、グラフウィンドウのサイズ変更がストレスなく可能。

比較のため、IRAFでもグラフを表示してみよう: `vocl> implot dev$pix`

- ・対話機能も利用可能。ただし、大文字のキー入力是一部未対応(現状でも)。

以下のキーが利用可能: C, I, R, T, U, ':'

例: --> `implot dev$pix`



メニューバー:

File: Print, Save, Loadなど。

Edit: Undo, Redo, Refresh,
Delete など。

Page: Next, Back, First, Last

Window: Newなど。

(複数のウィンドウを開くことが
可能)

Help: PyRAFグラフィックウィンドウ
のヘルプ

グラフの印刷方法

- ・グラフの印刷は、メニューの「File - Print」を選択するか、「=」キーの入力で可能。

(※実行環境に依存: プリンタ設定が正しく行われていることが前提)

- ・スクリプト内で印刷したい場合には、以下のように記述する。

```
from pyraf.gki import printPlot  
printPlot()
```

複数のグラフウィンドウの利用(1)

- ・PyRAFでは、複数のグラフウィンドウを同時に利用可能。

対話での操作:

メニュー Window – New で新規作成

Windowメニューでアクティブウィンドウを切り替え可能

消去: Quit Windowやウィンドウ右上の×ボタン。

複数のグラフウィンドウの利用(2)

スクリプト内の操作:

```
from pyraf import gwm
```

```
gwm.window('ウィンドウタイトル')
```

←「ウィンドウタイトル」のウィンドウをアクティブにする。
なければ新規作成。

ウィンドウの消去:

```
gwm.delete('ウィンドウタイトル')
```

→ 複数のグラフを同時に表示するスクリプトの作成が可能。

PyRAFでの画像描写

- displayタスクによるds9などへの画像描写や、imexamineなどの対話タスクによる操作も可能。
(imexamineによるグラフ描写は、**PyRAFカーネルの使用が条件**)
- ds9などへの画像描写は、IRAFカーネルを通してのみサポート。
PyRAF組み込みカーネルは開発中(恐らく今も)。

Pythonモードでの使用法

- ・IRAFタスク名の前に「iraf.」をつける。

例: `imstat` → `iraf.imstat()`

- ・タスクの引数は「,」で区切り、全体を括弧で括る。

文字列は「'」または「"」で囲む。Yes/noの略記の「+/-」は**使用不可**。

タスク名や変数名の**短縮は可能**

例: `imstat dev$pix for- fields=midpt`

→ `iraf.imstat('dev$pix', form=no, fi='midpt')`

Pythonモードでの使用法

- ・タスク名や変数名がPython予約語と一致する場合
「PY」をつける。

例: `lambda` → `iraf.PYlambda()`

- ・変数名の短縮形がPython予約語と一致する場合、
「PY」をつけるか、短縮形を止めるかのどちらでもよい。

例: `imcalc(in='filename')` → `iraf.imcalc(PYin='filename')`
or → `iraf.imcalc(inp='filename')`

Pythonモードでの使用例

```
iraf.displ('dev$pix',1)
```

```
iraf.imhe('dev$pix')
```

```
iraf.imhe('dev$pix', l=yes)
```

```
iraf.imstat('dev$pix')
```

```
iraf.imstat('dev$pix', form=no)
```

```
iraf.imstat('dev$pix', form=no, fi='mid')
```

→ 対話モードでも利用可能。試してみよう!

Pythonスクリプトでの記述

PyRAF:

2種類の使用方法:

- 対話モード
- Pythonモード

新しいタスクの作成:

IRAFタスクをPythonで書くための書式を知る必要あり。

タスクの実行や変数の設定方法は、複数の書式あり。

以下はあくまで一例。

PythonスクリプトでのPyRAFの使用

Pythonスクリプト内でPyRAFを使用:

1. pyraf.irasモジュールをimportする。(importの詳細は後述)

例: `from pyraf import iraf`

2. パッケージのロードやirasタスクの実行は Pythonモードでの記述をする。

例: `iras.daophot()`

`iras.imstat('dev$pix', form=no, fi='midpt')`

※ 引数なしでも「()」が必要。

PythonスクリプトでのPyRAFの使用

1'. パッケージやタスクの直接importも可能.

例: `from pyraf.iraf import phot`

2'. この場合、タスクやパッケージの「iraf.」が不要。

例: `iraf.phot(...)` → `phot(...)`

Pythonでのpipeの使用(1)

タスク変数「Stdin」、「Stdout」を使用する。

※ 変数名が大文字「S」から始まることに注意。

出力:

Stdout=1: 実行結果がリストとして変数に入る。

例: `s = iraf.imhead('dev$pix', long=yes, Stdout=1)`

入力:

Stdin=変数名: 変数の値を入力として受け取る。

例: `iraf.head(nl=3, Stdin=s)`

Pythonでのpipeの使用(2)

Stdin, Stdoutはファイル名やファイルハンドルも設定可能。

→ 入出力はファイルになる。

Stderrもリダイレクトとして利用可能。

タスク変数の設定:

幾つかの設定方法がある。

例1: 実行時に指定する。(タスクのデフォルト値は変わらず)

例1.1: `iraf.imcopy('dev$pix', 'mycopy.fits')`

例1.2: `iraf.imcopy(input='dev$pix', output='mycopy.fits')`

例2: 実行せず設定のみ行う(デフォルト値化):

`iraf.imcopy.input='dev$pix'`

`iraf.imcopy.output='mycopy.fits'`

`iraf.imcopy() # 実行`

タスク変数値の表示

変数値の表示も、幾つかの方法がある。

例1: IParamメソッドを使う:

```
iraf.imcopy.IParam()
```

例2: iraf.lparタスクを使う:

例2.1: `iraf.lpar(iraf.imcopy)`

例2.2: `iraf.lpar('imcopy')`

IRAFタスク実行の省力化

タイピングの負担を減らす方法:

irafモジュールの別名を定義する。

`i=iraf.daophot`

`i.datapars(...)`

`i.centerpars(...)`

`i.fitskypars(...)`

`i.phot(...)`

4. Python言語の基礎

Python言語の特徴(1)

- ・スクリプト言語: コンパイル不要
- ・オブジェクト指向言語: メソッド、クラス、継承あり。
- ・変数の定義(型宣言)が不要
- ・ブロックを(括弧ではなく)インデント(=字下げ)で表す。
 - 同じブロック内ではインデントの長さを揃える。
 - ブロック内でインデントが揃っていないとエラーとなる。
 - (ブロック最後の空行追加を推奨: なくてもエラーではないが可読性向上のため)
- ・拡張機能の利用:
 - 「import モジュール名」で必要な機能のみ追加可能。
 - # importの詳細は後述

Python言語の特徴(2)

- ・インクリメント・デクリメント演算子(++/--)がない。
→ 「+=」や「-=」で代用。 例: `i+=1`
- ・2.x系と3.x系があり、下位互換性がない。
→ 今回の実習: 2.x系を使用。(2.x系の最新版: 2.7.11)

例: 2.x系と3.x系の違い:

	2.x系	3.x系
print:	文	関数
モジュール名:	大文字始まり	小文字 (例: Tkinter vs tkinter)

拡張機能の利用: import (1)

Pythonは非常に多くの拡張機能を持つ。これらの機能は「import」でモジュールを取り込むことで利用可能となる。

importの用法: 以下の3つ。それぞれの例は次ページに掲載。

1. **import** モジュール名

「モジュール名」を取り込む。使用は、モジュール名.メソッド() など。

2. **import** モジュール名 **as** 別名

「モジュール名」を別名で取り込む。使用は、別名.メソッド() など。

3. **from** モジュール名 **import** サブモジュール名

「モジュール名」内のサブモジュールのみを取り込む。
使用は、サブモジュール名.メソッド()など。

拡張機能の利用: import (2)

importの使用例:

1. import モジュール名

```
import os  
os.remove(fn)
```

2. import モジュール名 as 別名

```
import matplotlib.pyplot as plt  
plt.show()
```

3. from モジュール名 import サブモジュール名

```
from praf import iraf  
iraf.imstat(fn)
```


データ型

Pythonのデータ型:

1. 整数
2. 小数
3. 文字列
4. 真偽(bool) yes or no
5. リスト
6. タプル: 変更不可なリスト。例: 関数の返り値
7. 辞書(dictionary) キー:値 ペアのリスト

整数・小数型

四則演算:

加: $a + b$

減: $a - b$

乗: $a * b$

除: a / b

べき乗: $x^{**}y$, `pow(x, y)`

リスト型

リスト型:

数値や文字列などを並べて格納できるデータ型。

書き方: それぞれの要素を「,」で区切って大括弧"[]"で囲む。

例: `data=['a', 'b', 'c']`

要素の指定:

0から始まる通し番号(=インデックス)をリスト型の変数名 + 大括弧[]で囲むことで指定可能。負の番号で後ろから要素を指定可能(-1: 最後の要素)。

例: `data[0] → 'a' (= data[-3])`

`data[1] → 'b' (= data[-2])`

`data[-1] → 'c' (= data[2])`

型変換

以下の関数を使用することで、データ型の変換が可能。

文字列型への変換: `str(変数)`

整数型への変換: `int(変数)`

小数型への変換: `float(変数)`

文字列操作(1)

- ・変数への代入: 「'」または「"」で囲む。

- ・文字列操作:

1. 結合: `str1 + str2` or `str.join([str1,str2])`

例: `'ABC' + 'DEF' → ABCDEF`

例: `'/'.join(['/home', 'hoge']) → /home/hoge`

(join: リストまたはタプルの要素を文字列strで結合した結果を返す)

2. 分割: `str.split(str2)`

文字列strを文字列str2で分割。結果をリストで返す。

例: `'02:35:41'.split(':') → ['02', '35', '41']`

文字列操作(2)

3. 文字列除去(先頭および末尾部分): `str.strip('chars')`

`str`文字列の先頭および末尾部分から、`chars`に該当した文字を削除した文字列を返す。`chars`を省略すると、空白(改行コード含む)を削除。

例:

```
'S0009400.fits¥n'.strip() → 'S0009400.fits'
```

```
'abcdefghijklmn'.strip('cbamln') → 'defghijk'
```

類似メソッドとして、以下の2つがある。

`str.lstrip('chars')` 先頭の文字列を削除

`str.rstrip('chars')` 末尾の文字列を削除

文字列操作(3)

4. 文字列の置換: `str.replace(old, new[,count])`

`str`文字列の中で部分文字列`old`を全て`new`に置換した文字列を返す。
オプション`count`を指定した場合、先頭から`count`個まで置換する。

例:

```
'S0009400.d.fits'.replace('d.fits','m.fits')
```

```
→ 'S0009400.m.fits'
```

```
'S0009400.n.fits'.replace('fits','cat')
```

```
→ 'S0009400.n.cat'
```

リスト操作(1)

リストのスライス:

`list[i:j]` インデックス `i` から `j-1` までの要素のみのリストを返す。

例: `a=[0,1,2,3,4]` の場合、 `a[1:3]` → `[1,2]`

`list[i:]` インデックス `i` から最後の要素までのリストを返す

例: `a=[0,1,2,3,4]` の場合、 `a[1:]` → `[1,2,3,4]`

`list[:j]` 最初の要素からインデックス `j-1` までの要素のリストを返す

例: `a=[0,1,2,3,4]` の場合、 `a[:2]` → `[0,1]`

リスト操作(2)

要素の追加:

`list.append(a)` リスト「list」の最後に要素「a」を追加する。

要素の削除:

`del list[i]` インデックス `i` の要素を削除

`list.pop(i)` インデックス `i` の要素を返し、リストから削除する。

`list.remove(a)` 要素 `a` を削除する。

引数の受け取り

スクリプト実行時に与えた引数:

sysモジュールのargv属性に文字列を要素とするリストとして格納。

sys.argvの使用には、sysモジュールのimportが必要。

リストの先頭要素sys.argv[0]はスクリプトファイル名(フルパスで)。

→ 引数を1個与えた場合、sys.argvの要素数は2になる。

使用法:

```
import sys          # sysモジュールのimport
print sys.argv[1:]  # 最初の要素を除く全要素を表示。
```

シェルコマンドの実行

subprocessモジュールのcallメソッドを使用する。
使用にはsubprocessモジュールのimportが必要。

使用法:

```
import subprocess  
subprocess.call(shellcommand, shell=True)
```

例:

```
import subprocess  
com='ds9&'  
subprocess.call(com, shell=True)
```

ファイルの存在確認と削除

ファイルの存在確認には、`os.path`モジュールの`os.path.isfile()`メソッドを、
ファイルの削除には、`os`モジュールの`os.remove()`メソッドを使用する。
使用には`os`モジュールの`import`が必要。

ファイルの存在確認: `os.path.isfile('ファイル名')` # IRAFの`access`と同等機能。

ファイルを削除: `os.remove('ファイル名')` # IRAFの`delete`と同等機能。

使用例: ファイル「myfile」が存在していたら削除:

```
import os                # osモジュールのimport
fn='myfile'
if os.path.isfile(fn):
    os.remove(fn)
```

ファイルサイズの確認

ファイルサイズの確認には、`os.path`モジュールの`os.path.getsize()`メソッドを使用する。返り値はバイト(B)単位。使用には`os`モジュールの`import`が必要。

ファイルの存在確認: `os.path.getsize('ファイル名')`

使用例: ファイル「myfile」が空ファイルであればスキップ:

```
import os                # osモジュールのimport
for fn in list:
    if os.path.getsize(fn) < 1: # ファイルサイズが1B未満であればスキップ。
        continue
```

ファイル読み書き(1)

1. ファイルオブジェクトの用意: open関数を使用する。

読み込み: `f = open('ファイル名', 'r')`

書き込み: `f = open('ファイル名', 'w')`

(補足: 追記:'a', 読み書き両用:'r+')

2. ファイルからの読み込み(推奨方法):

```
for line in f:
```

```
    print line,
```

(補足: 変数lineには改行も含まれるので
print文では「,」をつけて改行を抑制)

ファイルから1行ずつ読み出し、ループ内で処理をする。

(※ 変数lineには、改行コードも含まれるので注意。削除にはstripを使用)

ファイル読み書き(2)

3. ファイルへの書き込み:

`f.write(string)`

`string`の内容をファイルに書き込む。

文字列ではないものを書き込む場合、`str()`でまず文字列に変換する必要がある。

条件分岐: if文

if 条件式1:

 処理1 #インデント!

elif 条件式2:

 処理2 #インデント!

else:

 処理3 #インデント!

 # 空行(可読性向上)

処理 # インデント解除!

例:

```
if i > 5:
```

```
    print 'i > 5'
```

```
elif i < 0:
```

```
    print 'i < 0'
```

```
else:
```

```
    print '0 <= i <= 5'
```


繰り返し: for文

書式:

for 変数 in オブジェクト:

 処理 # インデント

シーケンス型のオブジェクトから要素をひとつずつ受け取り、変数に格納して処理を行う。これを最後の要素まで繰り返し実行する。

例:

```
list=['A', 'B', 'C', 'D']
```

```
for fn in list:
```

```
    print fn
```

比較・確認演算子

aとbは一致	<code>a == b</code>	
aとbは不一致	<code>a != b</code>	
aはbより大きい	<code>a > b</code>	
aはbより小さい	<code>a < b</code>	
aはb以上	<code>a >= b</code>	
aはb以下	<code>a <= b</code>	
aはbに含まれる	<code>a in b</code>	(bは文字列やリストなど)
aはbに含まれない	<code>a not in b</code>	(bは文字列やリストなど)

組み込み文: print (1)

print: 出力。

例: `print string` → 改行付きでstringの内容を表示。

`print string,` → 改行なしでstringの内容を表示。

文字列内で指定位置に変数の値を出力する方法: (Python 2.6以降)

'文字列'.format(変数1,変数2,...)と記述。どの変数をどの位置で出力するかは「{番号}」で制御。番号は0始まりで、0:変数1に対応。

```
print "a: {0} b:{1} c:{2} d:{3} e:{4}".format(A,B,C,D,E)
```

→ 'a: A b:B c:C d:D e:E'

```
print "a: {2} b:{1} c:{3} d:{4} e:{0}".format(A,B,C,D,E)
```

→ 'a: C b:B c:D d:E e:A'

組み込み文: print (2)

書式を制御した変数値の出力(簡略版): **{番号:[幅][.精度][タイプ]}**

書式指定子:

[幅]: フィールド幅(整数)

[.精度]: 小数点以下の最大桁幅(整数)

[タイプ]: s:文字列、d:10進数、f:固定小数点、e:指数指定、
g: 数値を有効桁「精度」で丸め、桁に応じてf/ eで表示。

例1: {0:7.5f} 1つめの変数の値を幅7の固定小数点表示(精度:5桁)で出力。

例2: 実例:	x=3.123456789	x=312.3456789	
{0:7.5f}.format(x)	3.12346	312.34568	
{0:7.5e}.format(x)	3.12346e+00	3.12346e+02	
{0:7.5g}.format(x)	3.1235	312.35	# 有効桁=5桁

組み込み文: print (3)

書式を制御した変数の値の出力(詳細版): **[整列][符号]番号[幅][.精度][タイプ]**

書式指定子:

[整列]: <:左詰、>:右詰、^:中央寄せ、=:0埋め(数値型のみ)

[符号]: +:正負数両方で符号表示、-:負数のみ符号表示(デフォルト)、
空白: 正数で空白、負数で符号表示。

[幅]: フィールド幅(整数)

[.精度]: 小数点以下の最大桁幅(整数)

[タイプ]: s:文字列、d:10進数、f:固定小数点、e:指数指定、
g: 数値を有効桁「精度」で丸め、桁に応じてf/eで表示。

詳細: <http://docs.python.jp/2/library/string.html#formatspec>

組み込み関数: len(), round()

len(): 長さを返す。

len(string): stringの内容の文字列のバイト数を返す

len(list): listの要素数を返す。

→ 引数sys.argvを要素数を確認する時に有用。

例: len(sys.argv)

round(値[,桁]):

「値」を小数点以下「桁」桁で丸めた浮動小数点の値を返す。

桁は省略可能で、省略すると小数点以下0桁で丸める。

例: round(0.5) → 1.0

round(-0.5) → -1.0

スクリプトの終了: sys.exit()

Pythonスクリプトの終了には、`sys.exit()`を使用する。

(補足: 組み込みの`exit()`でも終了できるが、こちらは対話用で、スクリプトでの使用を避けるべき。)

使用例:

条件を満たす場合にスクリプトを終了させる:

```
if (sys.argv != 3):           # 引数が2個以外の場合に、
    print 'Usage: ...'       # 用法を表示して
    sys.exit()               # スクリプトを終了。
```

関数の作成(1)

PyRAFを使用するPythonスクリプトでは、**関数の使用が必須**。

以下、関数の作成方法:

```
def 関数名(引数1, 引数2, ...):  
    処理... # インデント!
```

例:

```
def myfunc(list):  
    for f in list:  
        print f
```


関数の作成(2)

関数の引数にデフォルト値を設定することも可能。デフォルト引数(=IRAFの隠し引数に類似)は、**明示引数の後**で引数名=値と定義する。

```
def 関数名(明示引数1, 明示引数2, デフォルト引数1=デフォルト値):  
    処理... # インデント!
```

例: 明示引数list1, list2, デフォルト引数verb(デフォルト値=False)の場合:

```
def myfunc(list1, list2, verb=False): # デフォルト引数verbは明示引数の後。  
    for f in list1:  
        if verb == True: # デフォルト引数verb==Trueの場合に  
            print f # print f を実行。
```

関数の使用

関数を使う方法:

関数名(引数1, ...)

例:

<code>myfunc()</code>	# 引数なしの関数の実行
<code>myfunc(list)</code>	# 1つの明示引数を持つ関数の実行
<code>myfunc(list, a, b)</code>	# 3つの明示引数を持つ関数の実行
<code>myfunc(list, a, verb=True)</code>	# デフォルト引数の指定は「 引数=値 」
<code>myfunc(*list[1:4])</code>	# リスト名の前に「 * 」をつけると、リストを 展開して 関数の引数として与えられる。
<code>y = myfunc(list, a, b)</code>	# 関数の実行結果を変数yに代入

クラスの作成(例)

天体カタログの整理などの際に、複数のデータ(=属性)を1個の変数で保持できると便利。ここではそのような新しいデータ型(=クラス)を作成する。

以下は、2つの属性(=データ)を持つクラスを定義する例。属性の定義のみで、メソッドは定義していない。クラスの初期化時にのみ実行される特殊な関数「`__init__()`」を定義し、引数として受け取った値を属性に格納する(下線は「`_`」x2)。

```
class クラス名:                                # クラス名は()をつけない。
    def __init__(self, 引数1, 引数2):          # 初期化関数を定義。selfは必須。
        self.属性1 = 引数1                    # 引数1を属性1に格納。「self.」必須。
        self.属性2 = 引数2
```

クラスの使用(例)

使用法:

```
オブジェクト = クラス名(引数)    # インスタンスの作成  
print オブジェクト.属性1        # 属性1の表示
```

例:

```
class scat:                        # クラスscatの定義  
    def __init__(self, arg1, arg2): # 初期化関数の定義。self必須。  
        self.x = arg1  
        self.y = arg2  
  
a = scat(data1, data2)            # クラスscatのインスタンスaの作成  
print a.x, a.y                   # インスタンスaの属性x, yの表示
```

Pythonスクリプトの基本構文(1)

PyRAFを使用するPythonの基本構文:

```
#!/usr/bin/env python # 1
# -- coding: utf-8 -- # 2

import sys # 3
from pyraf import iraf # 4

def mytask(arg1,arg2): # 5
    iraf.imstat(arg1, format=no, fields=arg2) # 6
    ...

if __name__ == "__main__": # 7
    mytask(*sys.argv[1:3]) # 8
```

#1: シェルからスクリプトを実行するためのおまじない

#2: 文字コード設定: utf-8を使用

#3: sysモジュールのimport

#4: pyraf.irafモジュールのimport

#5: 「自作タスク」関数の定義

#6: irafタスクの記述

#7: シェルから実行された時のみ
#8を実行するためのおまじない

#8: 自作タスク実行の記述

Pythonスクリプトの基本構文(2)

補足:

- ・ #1 はシェルから実行しない、または sh スクリプト名 で実行する場合は不要。
- ・ #2 で coding でスクリプト内で使用する文字コードを宣言。ascii文字のみであれば宣言不要。
- ・ #5: Pythonで自作タスクを作成する場合、必ず「関数」でタスクを定義する。
- ・ #7: シェルから実行可能とする場合のみ必要。
(「__」は「_」(アンダーバー)2つ)
- ・ #8: 「*リスト名」で複数の要素を展開して関数に渡すことが可能。
つまり、`mytask(*sys.argv[1:3]) = mytask(sys.argv[1], sys.argv[2])`

Pythonスクリプトの実行

実行方法は2通り:

本実習では1を使用

1. シェルからコマンドとして実行:

```
$ スクリプト名 引数1 引数2 ... (パスを通してある場合)
```

2. PyRAF対話環境、Python対話環境で実行:

```
import スクリプトファイル幹名 (「.py」を除いた名前)
```

```
スクリプトファイル幹名.関数名(引数1, 引数2)
```

例: myscript.py内のmyfunc()の場合: (用法注意:

```
import myscript  
myscript.myfunc()
```

スクリプト置き場をPythonに登録するか、
import時にスクリプト置き場にcdで移動
しておく必要あり。詳細は割愛)

5. Pythonスクリプト実習

実習準備(1)

テキストでは、

1. スクリプトは ~/pylec/pyscripts/以下で作成する
2. スクリプトは データディレクトリ下で実行する

ことを前提とする。(スクリプト置き場とデータ置き場を分ける)

補足: データ置き場とスクリプト置き場を分ける理由:

スクリプトの管理・移植性の向上のため:

スクリプトは、大量の観測データ(観測日ごとにディレクトリが異なる)を処理することを想定している。スクリプトのみでまとめておく方が、管理(修正や更新など)や他環境への導入が容易。

実習準備(2)

スクリプト実行の省力化:

スクリプトを作る前に、スクリプト置き場にパスを通しておく(以下bashの例)。

(スクリプトをフルパスで指定せずに実行可能となる)

例: テキストエディタで ~/.bashrcを開き、末尾に

```
export PATH=$PATH:$HOME/pylec/pyscripts
```

を追記し、.bashrcを再読み込みしておく:

```
$. ~/.bashrc
```

補足: 再読み込み実行後に新しく開くターミナルでは .bashrcの再読み込みは

不要。既存のターミナルでは再読み込みが**必要**。

実習準備(3)

スクリプト編集用とデータ解析用にターミナルを二つ立ち上げる。

(既存のターミナルを使用する場合、ターミナル毎に「. ~/.bashrc」を実行する)

一つのターミナルはスクリプト置き場に移動しておく。

```
$ cd ~/pylec/pyscripts
```

もう一つのターミナルはデータディレクトリに移動しておく。

```
$ cd ~/pylec/data/100918
```

補足: スクリプトは実行前に実行権限を付与すること。

例: `$ chmod u+x スクリプトファイル名`

実習問題一覧

Pythonスクリプト実習として、以下の練習・実習問題を用意している:

0. 練習B0: 画像表示ソフト
1. 練習B1: dark作成ソフト
2. 実習B1: dark差し引きソフト
3. 練習B2: フラット作成ソフト
4. 実習B2: フラット補正ソフト
5. 練習B3: 天体検出ソフト
6. * 実習B3: 天体検出ソフトの改良
7. 練習B4: 画像マッチングソフト
8. * 実習B4: 画像マッチングソフトの改良
9. 練習B5: アパーチャ測光ソフト
10. 練習B6: 比較測光ソフト
11. * 実習B6: 比較測光ソフトの改良
12. * 実習B7: 100925の処理

補足:「*」付きの実習は「時間に余裕のある人」向け。飛ばしても最後まで実習(+処理)が可能。(ただし、改良版は結果がわずかに異なる)

練習B0: 画像表示ソフトの作成

CLスクリプト実習の練習A1で作成したソフトと同じ機能を持つソフトをPythonで作成する。

機能: フレーム名のリストを受け取り、3秒間隔でフレームを表示する

ソフト名: `mdisp1.py`

使用タスク: `sections`, `display`, `sleep` (練習A1と同じ)

実行例: `mdisp1.py @object.list`

(補足: 実行前にスクリプトファイルに実行権限を付与する:
例: `$ chmod u+x mdisp1.py`)

練習B0のスクリプト例:

~/pylec/samples/pyscripts/mdisp1(_c).py (*_c.py:コメント付)

```

mdisp1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
#!/usr/bin/env python
# -- coding: utf-8 --
# irafのdisplay, sleepを用いて複数枚の画像を3秒おきにds9に表示する。
# ds9の起動が前提。スクリプト内での起動チェックは未実装。
# M.Isogai
# ver.1 mdisp1.cと同じ機能。

# sysモジュールのimport。 引数を使用するため。
import sys
# osモジュールのimport。 ファイルの存在確認 os.path.isfile()と
# ファイル削除 os.remove() を使用するため。
import os
# pyraf.irafモジュールのimport。 pythonでirafを使用するため。
from pyraf import iraf

# 関数「mdisp1」の定義:
def mdisp1(images):

    # 一時ファイルの作成。
    infn=iraf.mktemp("tmp_") # use iraf.mktemp

    # sectionsを用いて入力ファイルまたはリストを一時ファイルに格納:
    iraf.sections(images, option="fullname", Stdout=infn)

    # 一時ファイルを開く。
    fp=open(infn,"r")
    # 入力リストから1行ずつ読み込み、最後の行まで繰り返す。
    for img in fp:

        # 行頭や行末の空白(改行コード含む)を削除
        img=img.strip()

        # 表示するフレーム名を出力する。
        print "display ", img

```

```

mdisp1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
print "display ", img

# irafのdisplayタスクで画像を表示。
iraf.display(img,1)

# irafのsleepタスクで3秒のsleepをかける。
iraf.sleep(3)

# 一時ファイルの削除:
os.remove(infn)

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

    # 用法の表示。引数の数が1以外では用法を出力し、スクリプトを終了する。
    # リストsys.argvにはスクリプトファイル名(フルパスで)と引数が格納されている。
    # スクリプト実行時に引数を1個指定した場合、リストの要素数は2個になる。
    if len(sys.argv) != 2:
        print "Usage: {0} [fits files or @input-list]".format(sys.argv[0])
        print "cf: {0} S00*.fits".format(sys.argv[0])
        print "cf: {0} @object.list".format(sys.argv[0])
        sys.exit()

    # スクリプト実行時の引数1個を与え、関数「mdisp1」を実行する。
    mdisp1(sys.argv[1])

```

解析ソフト作成について

これから、実際の観測データを解析するソフトウェアをPythonで作成していく。観測データは撮像観測のデータで、比較測光結果の表示まで用意している。処理の流れは以下の通り。

1. 一次処理その1 (ダーク作成、差し引き)
2. フラット補正画像の作成
3. 一次処理その2 (フラット補正)
4. 天体検出
5. 画像マッチング
6. アパーチャ測光
7. 比較測光

使用するIRAFのタスク(とファイル名)

- | | # 使用するファイル名 |
|--|------------------------------|
| 1. 一次処理 その1: | |
| 1.1 ダーク作成: imstat, imcombine | |
| 1.2 ダーク差し引き: imarith | *.fits → *.d.fits |
| 2. フラット補正画像の作成: mktemp, imarith, imstat, imcombine | |
| 3. 一次処理 その2 (フラット補正): imcombine | *.d.fits → *.n.fits |
| 4. 天体検出: 外部コマンド「SExtractor」 | *.n.fits → *.n.cat |
| 5. 画像マッチング: | |
| 5.1 天体マッチング: xyxymatch | *.n.cat → *.n.cat.xyxymatch |
| 5.2 シフト量測定: geomap | *.xyxymatch → *.n.cat.geomap |
| 5.3 画像シフト: geotran | *.n.fits → *.s.fits |
| 6. アパーチャ測光: daophot.phot | *.s.fits → *.s.fits.mag.1 |
| 7. 比較測光: daophot.pdump | *.mag.1 → outphot.dat |

テストデータ

~/pylec/data/

→ 100918, 100925, 100929 の3つ。

天体の観測データ: 100918, 100925の2日分 **(まずは100918を処理)**

フラット用データ: 100929

各ディレクトリ内のファイル:

S??????.fits: 撮像データ

obs_Scam.list: 観測ログ

.list: 各種フレームリスト(object.list, dark.list, flat*.list, ...)

練習B1 dark作成ソフト

darkフレームリストを受け取って、各フレームの統計値を表示し、medianダークフレームを作成するソフト「mkdark.py」を作る。

ダークフレームを作成する前に、すでに同名のファイルが存在する場合には削除する。引数の数が正しくない場合にはヘルプを表示し、リストは「@」なしで与えるものとする(以降も共通の仕様とする)。

使用タスク: imstat, imcombine

実行: `$ mkdark.py ダークリスト名 出力フレーム名`

実行例: `mkdark.py dark.list dark.fits`

(実行前の実行権限の付与を忘れないこと: `$ chmod u+x mkdark.py`)

medianダークフレーム作成ソフトの例

~/pylec/samples/pyscripts/mkdark(_c).py (*_c.py:コメント付)

```

mkdark_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
#!/usr/bin/env python
# -- coding: utf-8 --
# irafのimcombineを用いて複数枚のダークフレームからmedianフレームを作成する。
# M.Isogai

# sysモジュールのimport 引数を使用するため。
import sys
# osモジュールのimport。ファイルの存在確認 os.path.isfile()と
# ファイル削除 os.remove() を使用するため。
import os
# pyraf.irafモジュールのimport pythonでirafを使用するため。
from pyraf import iraf

# 関数「mkdark」の定義:
def mkdark(list,ofn):

    # 出力フレーム(ofn)がすでに存在しているか確認(os.path.isfile)し、
    # 存在していれば消去(os.remove)する。
    if os.path.isfile(ofn):
        os.remove(ofn)

    # 入力リスト名にリストを表すワイルドカード「@」を追加。
    list = '@' + list

    # imstatを実行し、結果を出力。リスト内のフレームチェック用。
    iraf.imstat(list,format="no", fields="image,mean,midpt")

    # imcombineを実行。medianフレーム作成。
    iraf.imcombine(list,ofn, combine="median", outtype="real")

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

    # 用法の表示。引数の数が2以外では用法を出力し、スクリプトを終了する。

```

```

mkdark_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
# 用法の表示。引数の数が2以外では用法を出力し、スクリプトを終了する。
# sys.argv[0]にはスクリプトファイル名が格納されるため、引数が2個の場合、
# リストsys.argvの要素数は3になる。
if len(sys.argv) != 3:
    print "Usage: {0} [input-list] [output-frame-name]".format(sys.argv[0])
    print "cf: {0} dark.list dark.fits".format(sys.argv[0])
    sys.exit() # スクリプトの終了

# 関数「mkdark」を実行する。スクリプト実行時の引数を関数に与える。
# sys.argvはスクリプト実行時の引数が格納されているリスト。
# 1番目の要素はスクリプトファイル名(フルパス)が格納されるため、
# 2つの引数へは、2番目の要素と3番目の要素を指定する必要がある。
# *リスト名とリスト名の前に「*」をつけることで、リストの要素を展開して
# 関数に渡すことが可能。
# リスト名[1:3] = リスト名[1], リスト名[2]
mkdark(*sys.argv[1:3])

```

実習B1: ダーク差し引きソフトの作成

入力リスト中のフレームに対してダークを差し引くソフト「subdark.py」を作る。
ヘルプの表示条件とリストの与え方は練習B1と共通(以降も共通)。

使用タスク: imarith

実行: subdark.py 入力リスト ダークフレーム名 出力リスト

実行例: `subdark.py object.list dark.fits object.d.list`

ダーク差し引きソフトの例

~/pylec/samples/pyscripts/subdark(_c).py (*_c.py:コメント付)

```

subdark_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
#!/usr/bin/env python
# -- coding: utf-8 --
# irafのimarithを用いて生データフレームからダークフレームを差し引く。
# M.Isogai

# sysモジュールのimport 引数を使用するため。
import sys
# pyraf.irafモジュールのimport pythonでirafを使用するため。
from pyraf import iraf

# 関数「subdark」の定義:
def subdark(inlist,dfn,ofn):

    # モニタにメッセージを表示。{0},{1},{2}の位置に「.format()」で指定した
    # 変数(inlist, dfn, ofn)の値が表示される。
    # {}内の数字と「.format()」メソッドに与えた引数との対応関係は、以下の通り:
    # {0} = inlist, {1} = dfn, {2} = ofn
    print "# subtract dark: {0} - {1} = {2}".format(inlist,dfn,ofn)

    # 入力フレームリスト名、出力フレームリスト名にリストを表すワイルドカード「@」を
    # 追加。
    inlist = '@' + inlist
    ofn = '@' + ofn

    # imarithを実行。ダーク差し引き: @inlist - dfn = @ofn
    iraf.imarith(inlist,'-',dfn, ofn, title="",divzero=0, hparams="",\
                pixtype="",calctype="",verbose='no',noact='no')

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

    # 用法の表示。引数の数が3以外では用法を出力し、スクリプトを終了する。

```

```

subdark_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
# 用法の表示。引数の数が3以外では用法を出力し、スクリプトを終了する。
# 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
if len(sys.argv) != 4:
    print "Usage: {0} [input-list] [dark-frame] [output-list]".format(sys.a
argv[0])
    print "cf: {0} object.list dark.fits object.d.list".format(sys.argv[0
])
    sys.exit() # スクリプトの終了

# スクリプト実行時の3個の引数を与え、関数「subdark」を実行する。
# 「*sys.argv[1:4]」の詳細についてはmkdark_c.pyのコメントを参照。
subdark(*sys.argv[1:4])

```

練習B2の下準備

練習B2のフラット作成ソフトの作成を始める前に、フラットデータディレクトリに移動し、ダーク作成とダーク差し引きを実行しておく:

例:

```
$ cd ../100929 (100918にいた場合)
```

```
$ mkdark.py dark.list dark.fits
```

```
$ subdark.py flat.list dark.fits flat.d.list
```

練習B2: フラット補正フレーム作成ソフト

ダーク差し引き済みフレームからフラット補正用フレームを作成するソフト「mkflat.py」を作る。

入力フレームのmedianを計算し、その値で入力フレームの規格化を行ってから(*.d.fits → *.m.fits)、全フレームをmedian結合する。mktempを一時フレームリストの作成に使用する。結合前に同名のフレームが存在していれば削除する。一時ファイルの削除にはosモジュールのremoveメソッドを使用する。

使用タスク: mktemp, imstat, imcombine, imarith

実行: mkflat.py 入力リスト 出力フレーム

実行例: `mkflat.py flat.d.list flat.fits`

フラット補正フレーム作成ソフトの例(1)

~/pylec/samples/pyscripts/mkflat(_c).py (*_c.py:コメント付)

```

mkflat_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

#!/usr/bin/env python
# -- coding: utf-8 --
# ダーク差し引き済みフラットフレームリストからフラットフィールドング用フレームを
# 作成する。
# 処理: 1:入力各フレームのメジアンを計算。そのメジアンで割る。
#       2: メジアンで割ったフレームたちをメジアン結合(=出力フレーム)。
# M. Isogai

# sysモジュールのimport。 引数を使用するため。
import sys
# osモジュールのimport。ファイルの存在確認 os.path.isfile()と
# ファイル削除 os.remove() を使用するため。
import os
# pyraf.irafモジュールのimport。 pythonでirafを使用するため。
from pyraf import iraf

# 関数「mkflat」の定義:
def mkflat(inlist, ofn):

    # 2つの一時ファイル名を作成:
    tmpf1=iraf.mktemp('tmp_')
    tmpf2=iraf.mktemp('tmp_')

    # 各フレームのメジアンを計算し、結果を一時ファイル「tmpf1」に出力。
    print "# find the median..."
    iraf.imstat('@' + inlist, fields="midpt", lower='INDEF', \
               upper='INDEF', nclip=2, lsigma=3., usigma=3., \
               binwid=0.1, format='no', cache='yes', Stdout=tmpf1)

    ### メジアンで割ったフレーム名の作成
    # 入力リストと一時ファイルを開く。
    fp1=open(inlist, 'r')
    fp2=open(tmpf2, 'w')

```

ページ1/3

```

mkflat_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

fp2=open(tmpf2, 'w')

# 入力リストから1行ずつ読み込み、最後の行まで繰り返す。
for f in fp1:
    # フレーム名の拡張子「d.fits」を「m.fits」に置換
    g=f.replace('d.fits', 'm.fits')
    # 置換したフレーム名を一時ファイル(tmpf2)に書き込み。
    fp2.write(g)

# ファイルを閉じる。書き込みの場合、閉じて初めてファイルがHDD上に生成される。
# ここで閉じておかないと、一時ファイルを後ろでオープン出来ない。
fp1.close()
fp2.close()

# 各フレームをそれぞれのメジアンで規格化したフレームを作成:
print "# normalize..."
iraf.imarith('@' + inlist, '/', '@' + tmpf1, '@' + tmpf2, \
            title="", divzero=0, hparams="", \
            pixtype="", calctype="", verbose='no', noact='no')

# チェックのため、メジアンで規格化したフレームたちの統計値を計算:
iraf.imstat('@' + tmpf2, fields="image, mean, midpt", lower='INDEF', \
           upper='INDEF', nclip=2, lsigma=3., usigma=3., \
           binwid=0.1, format='no', cache='yes')

# 出力フレーム(ofn)がすでに存在しているか確認(os.path.isfile)し、
# 存在していれば削除(os.remove)する。
if os.path.isfile(ofn):
    os.remove(ofn)

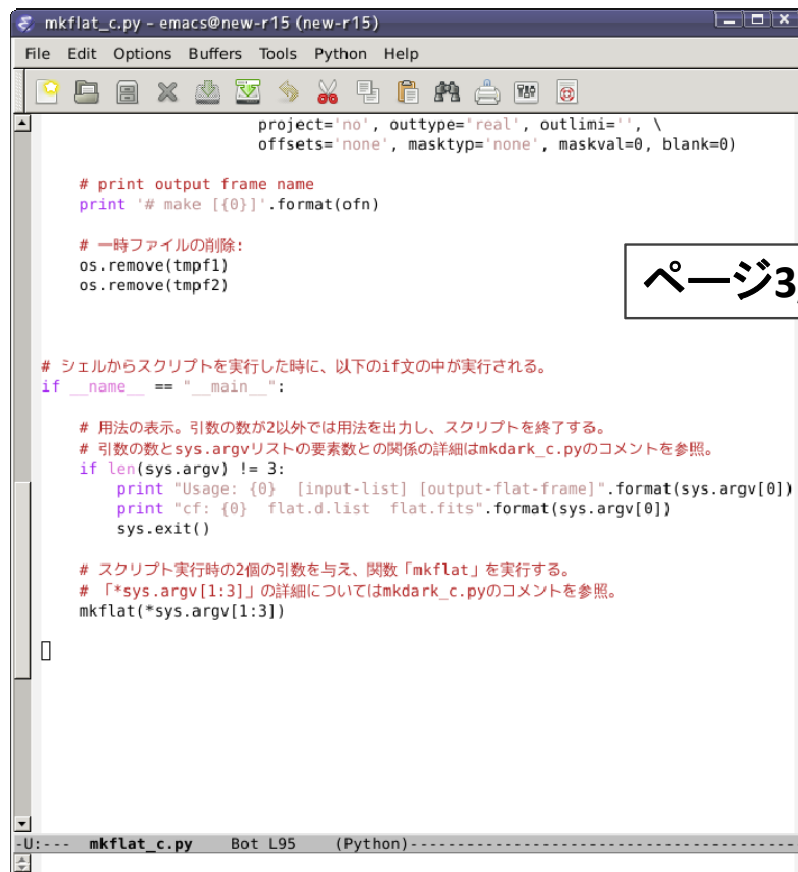
# 各メジアンフレームをimcombineでメジアン結合:
print "# combine..."
iraf.imcombine('@' + tmpf2, ofn, combine='median', reject='none', \
              project='no', outtype='real', outlimi='', \
              offset='input', result='input', result2='input', result3='input', result4='input', result5='input', result6='input', result7='input', result8='input', result9='input', result10='input', result11='input', result12='input', result13='input', result14='input', result15='input', result16='input', result17='input', result18='input', result19='input', result20='input', result21='input', result22='input', result23='input', result24='input', result25='input', result26='input', result27='input', result28='input', result29='input', result30='input', result31='input', result32='input', result33='input', result34='input', result35='input', result36='input', result37='input', result38='input', result39='input', result40='input', result41='input', result42='input', result43='input', result44='input', result45='input', result46='input', result47='input', result48='input', result49='input', result50='input', result51='input', result52='input', result53='input', result54='input', result55='input', result56='input', result57='input', result58='input', result59='input', result60='input', result61='input', result62='input', result63='input', result64='input', result65='input', result66='input', result67='input', result68='input', result69='input', result70='input', result71='input', result72='input', result73='input', result74='input', result75='input', result76='input', result77='input', result78='input', result79='input', result80='input', result81='input', result82='input', result83='input', result84='input', result85='input', result86='input', result87='input', result88='input', result89='input', result90='input', result91='input', result92='input', result93='input', result94='input', result95='input', result96='input', result97='input', result98='input', result99='input', result100='input')

```

ページ2/3

フラット補正フレーム作成ソフトの例(2)

~/pylec/samples/pyscripts/mkflat(_c).py (*_c.py:コメント付)



```
mkflat_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

project='no', outtype='real', outlimi='', \
offsets='none', masktyp='none', maskval=0, blank=0)

# print output frame name
print '# make [{0}]'.format(ofn)

# 一時ファイルの削除:
os.remove(tmpf1)
os.remove(tmpf2)

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

# 用法的表示。引数の数が2以外では用法を出力し、スクリプトを終了する。
# 引数の数と sys.argv リストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
if len(sys.argv) != 3:
    print "Usage: {0} [input-list] [output-flat-frame]".format(sys.argv[0])
    print "cf: {0} flat.d.list flat.fits".format(sys.argv[0])
    sys.exit()

# スクリプト実行時の2個の引数を与え、関数「mkflat」を実行する。
# 「*sys.argv[1:3]」の詳細についてはmkdark_c.pyのコメントを参照。
mkflat(*sys.argv[1:3])

-U:--- mkflat_c.py Bot L95 (Python)-----
```

ページ3/3

実習B2: フラット補正ソフト

練習2で作成したフラット補正用フレームでフラットフィールドイングを行うソフト「corff.py」を作る。

使用タスク: imarith

実行: corff.py 入力リスト フラット補正フレーム 出力リスト

実行例: corff.py object.d.list ../100929/flat.fits object.n.list

※ 100918ディレクトリに移動してから実行する。

例: \$ cd ../100918

フラット補正ソフトの例

~/pylec/samples/pyscripts/corff(_c).py (*_c.py:コメント付)

```
corff_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
#!/usr/bin/env python
# -- coding: utf-8 --
# 入力リスト中のフレームに対してフラットフィールドングを実施する。
# M.Isogai

# sysモジュールのimport。 引数を使用するため。
import sys

# pyraf.irafモジュールのimport。 pythonでirafを使用するため。
from pyraf import iraf

# 関数「corff」の定義:
def corff(inlist,ff,outl):

    # 入力引数と実行処理の情報を出力:
    print "# flat filed correction: {0} / {1} = {2}".format(inlist,ff,outl)

    # imarithを実施: '@'+inlist / ff = '@'+outl
    iraf.imarith('@' + inlist, '/', ff, '@' + outl, title="",divzero=0, \
        hparams="", pixtype="", calctype="", verbose='no', \
        noact='no')

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

    # 用法の表示。引数の数が3以外では用法を出力し、スクリプトを終了する。
    # 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
    if len(sys.argv) != 4:
        print "Usage: {0} [input-list] [flat-fielding frame] [output-list]".format(sys.argv[0])
        print "cf: {0} object.d.list ../100929/flat.fits object.n.list".format(sys.argv[0])
        sys.exit()
```

```
corff_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
print "cf: {0} object.d.list ../100929/flat.fits object.n.list".format(sys.argv[0])
sys.exit()

# スクリプト実行時の3個の引数を与え、関数「corff」を実行する。
# 「*sys.argv[1:4]」の詳細についてはmkdark_c.pyのコメントを参照。
corff(*sys.argv[1:4])

-U:--- corff_c.py Bot L40 (Python) -----
Mark set
```

練習B3: 天体検出ソフト

外部の天体検出ソフト「Source Extractor」を使用して、天体を検出するソフト「fstar1.py」を作る。

→ Source Extractorについては次ページを参照。

Source Extractorはシェルコマンドとして実装されているため、Pythonスクリプトからの実行にはsubprocessモジュールのcallメソッドを使用する。

使用タスク: なし

実行: fstar1.py 入力リスト

実行例: `fstar1.py object.n.list`

Source Extractorについて

「Source Extractor」: 銀河検出用に開発されたソフト。星の検出にも使える。

本家HP: <http://www.astromatic.net/software/sextractor>

ドキュメント: <https://www.astromatic.net/pubsvn/software/sextractor/trunk/doc/sextractor.pdf>

Source Extractorの実行方法:

`sex 入力フレーム -c confファイル -CATALOG_NAME カタログ名 ¥`

`-PARAMETERS_NAME 変数ファイル名 -FILTER N -VERBOSE_TYPE QUIET`

(スクリプト中では1行で書く!)

confファイル: 設定ファイル。用意済み(次ページを参照)。

変数ファイル: 出力変数(=情報)を指定するファイル。用意済み(次ページを参照)。

カタログ名: 入力フレームを元に作成: *.fits → *.cat **(フレーム毎に異なる)**

Source Extractor 出力変数一覧

サンプルの変数ファイルで出力を指定している変数は以下の通り:

- 1, 2: X_IMAGE, Y_IMAGE: 検出天体の中心座標
- 3: NUMBER: 通し番号
- 4: FWHM_IMAGE: 検出天体の半値幅
- 5, 6: FLUX_AUTO, FLUXERR_AUTO: 検出天体のFLUX(=総ADU値)とその誤差
- 7, 8: MAG_AUTO, MAGERR_AUTO: 検出天体の機械等級と誤差
- 9: FLUX_MAX: ピークカウント値
- 10: BACKGROUND: バックグラウンド値
- 11: THRESHOLD: 天体検出の閾値
- 12: FLGAS: 検出フラッグ: 0以外の場合、何らかの問題あり

「AUTO」、FLAGの詳細は割愛。
Source Extractorのドキュメントを参照

天体検出ソフトの補足

サンプルスクリプトでは、二つの関数を用意し、リストに対する処理と各フレームに対する処理を分離している。後者の関数にはデフォルト引数verbを設定し(デフォルト値=False)、verb=Trueの場合は動作確認を兼ねてSource Extractorの実行コマンドを表示する。

Source Extractorのconfファイルと変数ファイルは、以下に用意済み:

confファイル: `~/pylec/samples/pyscripts/sex_find.conf`

変数ファイル: `~/pylec/samples/pyscripts/sex_find.param`

サンプルスクリプトでは、スクリプトファイル名を元に置き場のパスを作成し、それぞれのファイル名と結合して、フルパスでのファイル名を作成している。

天体検出ソフトの例(1)

~/pylec/samples/pyscripts/fstar1(_c).py (*_c.py:コメント付)

```
fstar1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

#!/usr/bin/env python
# -- coding: utf-8 --
# 外部ソフトウェア「Source Extractor(コマンド名:sex)」を使用して天体検出を行う。
# M.Isogai

# sysモジュールのimport  引数を使用するため。
import sys
# subprocessモジュールのimport。シェルコマンド(sex)実行のため。
import subprocess

# 関数「findstar」の定義:
#  引数=(フレーム名、設定ファイル名、出力カタログ名、
#  出力パラメータ指定ファイル名、出力情報制御フラッグ)
def findstar(image,conff,catn,parf,verb=False):
    # オプション引数verbがTrueならばif文内を実施。
    if verb == True:
        print "# find star with SExtractor: {0} => {2}".format(image,catn)

    # シェルコマンド
    com="sex {0} -c {1} -CATALOG_NAME {2} -PARAMETERS_NAME {3} -FILTER N -VERBOS
E_TYPE QUIET".format(image,conff,catn,parf)

    # 実行コマンドを表示:
    print "com: {0}".format(com)

    # シェルコマンドの実行。subprocess.call()メソッドを使用。
    subprocess.call(com,shell=True)

# 関数「findstarall」の定義。この中で関数findstarを実行。
#  引数=(スクリプトファイル名(フルパスで)、入力リスト名)
def findstarall(com,list):

    ### 設定ファイル名conffと出力パラメータ指定ファイル名parfを設定。
-U:--- fstar1_c.py Top L34 (Python)-----
```

ページ1/3

```
fstar1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

### 設定ファイル名conffと出力パラメータ指定ファイル名parfを設定。
### 両者はフルパスで指定する必要あり。
### スクリプト置き場と同じ場所に置いてあるため、スクリプトファイル名を
### 元に両ファイル名を作成する。
# スクリプトファイル名を「/」区切りで分割。
tmplist=com.split('/') # 分割結果はリストで返す。
del tmplist[-1] # リストの最後の要素(=ファイル名のみ)を削除。
dir = '/'.join(tmplist) # リストの全要素を「/」で結合。
conff= dir + "/sex_find.conf" # 設定ファイル名を追加。
parf= dir + "/sex_find.param" # パラメータ制御ファイル名を追加。

# 入力リストを読み込みで開く。
fp=open(list,'r')
# 一行ずつ読み込み、ブロック内でfindstar関数を実行する。
for f in fp:
    f=f.strip() # 行頭や行末の空白・タブ・改行コードを削除
    g=f.replace('fits','cat') # 入力フレーム名を元に出力カタログ名を作成
    findstar(f,conff,g,parf) # 関数findstarを実行。

# ファイルを閉じる。
fp.close()

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

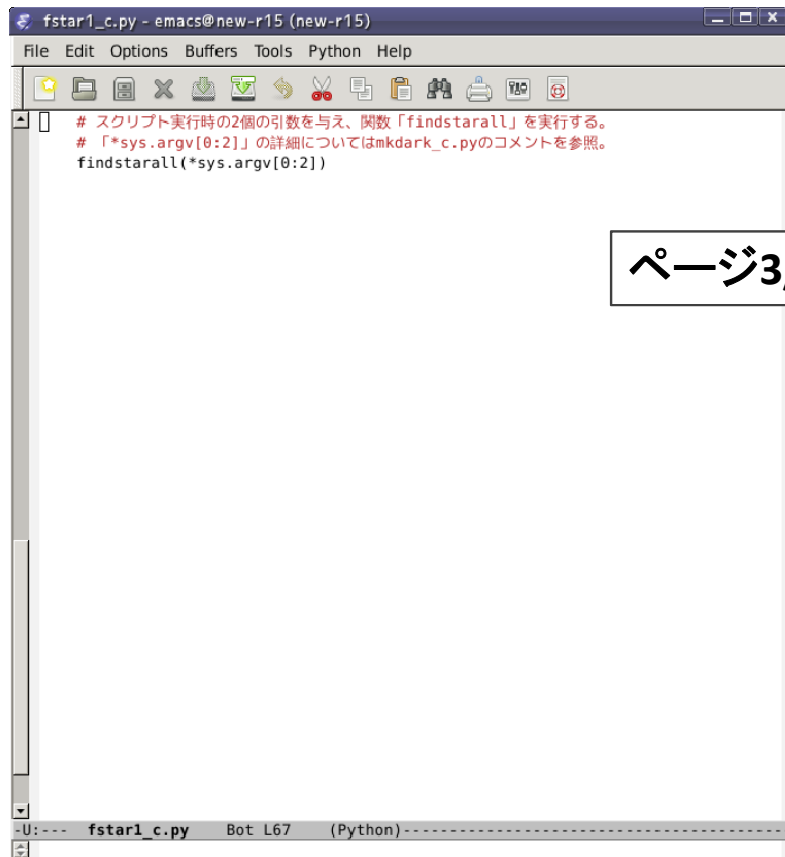
    # 用法の表示。引数の数が1以外では用法を出力し、スクリプトを終了する。
    # 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
    if len(sys.argv) != 2:
        print "Usage: {0} [input-list]".format(sys.argv[0])
        print "cf: {0} object.n.list".format(sys.argv[0])
        sys.exit()

    # スクリプト実行時の2個の引数を与え、関数「findstarall」を実行する。
    # 「*sys.argv[0:2]」の詳細についてはmkdark_c.pyのコメントを参照。
-U:--- fstar1_c.py 42% L67 (Python)-----
```

ページ2/3

天体検出ソフトの例(2)

~/pylec/samples/pyscripts/fstar1(_c).py (*_c.py:コメント付)



```
fstar1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
# スクリプト実行時の2個の引数を与え、関数「findstarall」を実行する。
# 「*sys.argv[0:2]」の詳細についてはmkdark_c.pyのコメントを参照。
findstarall(*sys.argv[0:2])

-U: --- fstar1_c.py Bot L67 (Python)-----
```

ページ3/3

実習B3: 天体検出ソフトの改良(1)

練習B3で作成した天体検出ソフトが出力するカタログファイルには偽天体も含まれている。これら偽天体を排除し、以下の判定条件・制限条件を満たす天体のみカタログに出力するようソフトを改良する(fstar2.py)。

判定条件: 半値幅 < 3.0 pix (~ 1 arcsec) または $\text{flag} > 0$ は偽天体として除外。

リストの制限条件: 天体数 ≤ 15 and 最も明るい天体との等級差 < 5.0 mag

※ 余裕がある受講生向け

実習B3: 天体検出ソフトの改良(2)

補足: サンプルスクリプトでは、カタログの編集用の関数を用意し、Source Extractor実行後、その関数を呼び出している。カタログ編集では、

- ・ 天体検出結果を格納するクラスscatを作成。
- ・ 検出天体のインスタンスを作成。判定条件を満たす天体のみリストに追加。
- ・ このリストを明るい順に並び替え。

並び替えにはsorted()関数とlambda式を組み合わせ使用。

例: `b = sorted(a, key=lambda a:float(a.mag))`

a: クラスscatのインスタンスのリスト

a.mag: インスタンスaの等級

実習B3: 天体検出ソフトの改良(3)

- ・ 並び替え後、制限条件を満たす天体のみで出力用のリストを作成。
- ・ 元のカatalogファイルをシェルコマンド「mv」で別名「*.org」にする。
同名のファイルが存在していれば削除。
- ・ リストの内容をファイルへ書き込み。

の処理を実装している。

天体検出ソフトの改良例(1)

~/pylec/samples/pyscripts/fstar2(_c).py (*_c.py:コメント付)

```
fstar2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
#!/usr/bin/env python
# -- coding: utf-8 --
# 外部ソフトウェア「Source Extractor(コマンド名:sex)」を使用して天体検出を行う。
# ver.2 検出結果の並べ替えとセレクションの処理を追加。
# M.Isogai

# sysモジュールのimport。 引数を使用するため。
import sys
# osモジュールのimport。 ファイルの存在確認(os.path.isfile)と削除(os.remove)。
import os
# subprocessモジュールのimport。 シェルコマンド(sex)実行のため。
import subprocess

# 関数「findstar」の定義:
# 引数=(フレーム名、設定ファイル名、出力カタログ名、
#       出力パラメータ指定ファイル名、出力情報制御フラッグ、
#       出力結果のチェックフラッグ)
def findstar(image, confff, catn, parf, verb=False, fchkat=True):
    # オプション引数verbがTrueならばif文内を実施。
    if verb == True:
        print "# find star with SExtractor: {0} => {2}".format(image, catn)

    # シェルコマンド
    com="sex {0} -c {1} -CATALOG_NAME {2} -PARAMETERS_NAME {3} -FILTER N -VERBOS
E_TYPE QUIET".format(image, confff, catn, parf)

    # 実行コマンドを表示:
    print "com: {0}".format(com)

    # シェルコマンドの実行。subprocess.call()メソッドを使用。
    subprocess.call(com, shell=True)

    # 出力結果のチェックフラッグが真の場合、関数「chkcat」を実行する。
    if fchkat == True:
-U:--- fstar2_c.py Top L34 (Python) ---
```

ページ1/6

```
fstar2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
if fchkat == True:
    chkcat(catn,owflag=False)

# 関数「findstarall」の定義。この中で関数findstarを実行。
# 引数=(スクリプトファイル名(フルパスで)、入力リスト名)
def findstarall(com, list):

    ### 設定ファイル名confffと出力パラメータ指定ファイル名parffを設定。
    ### 両者はフルパスで指定する必要あり。
    ### スクリプト置き場と同じ場所に置いてあるため、スクリプトファイル名を
    ### 元に両ファイル名を作成する。
    # スクリプトファイル名を「/」区切りで分割。
    tmplist=com.split('/') # 分割結果はリストで返す。
    del tmplist[-1] # リストの最後の要素(=ファイル名のみ)を削除。
    dir = '/'.join(tmplist) # リストの全要素を「/」で結合。
    confff= dir + "/sex_find.conf" # 設定ファイル名を追加。
    parff= dir + "/sex_find.param" # パラメータ制御ファイル名を追加。

    # 入力リストを読み込みで開く。
    fp=open(list,'r')
    # 一行ずつ読み込み、ブロック内でfindstar関数を実行する。
    for f in fp:
        f=f.strip() # 行頭や行末の空白・タブ・改行コードを削除
        g=f.replace('fits','cat') # 入力フレーム名を元に出力カタログ名を作成
        findstar(f,confff,g,parff) # 関数findstarを実行。

    # ファイルを閉じる。
    fp.close()

# 関数「chkcat」の定義:
def chkcat(catn,owflag=True):
    # 検出結果のセレクションの条件を指定する:
-U:--- fstar2_c.py 19% L67 (Python) ---
```

ページ2/6

天体検出ソフトの改良例(2)

~/pylec/samples/pyscripts/fstar2(_c).py (*_c.py:コメント付)

```
fstar2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
def chkcat(catn,owflag=True):
    # 検出結果のセレクションの条件を指定する:
    fwhm_limit=3.0 # lower limit of fwhm [pix] (corresponding to about 1")
    diffmag_limit=5.0 # upper limit of differential magnitude
    nmax=15 # maximum number of stars to output
    # 入力ファイルを読み込みで開く。
    fp=open(catn,'r')
    # リストを初期化:
    a=[]
    out=[]
    # 一行ずつ読み込み、ブロック内で処理をする。
    for f in fp:
        # 行頭の文字が「#」の場合、リストoutに追加して、それ以降の処理を省いて
        # 次の繰り返しへ。
        if f[0] == '#':
            out.append(f)
            continue
        # クラスscatのインスタンスbを作成する。
        b=scat(f)
        # 半値幅が下限値未満の場合、それ以降の処理を省いて次の繰り返しへ。
        if b.fwhm < fwhm_limit:
            continue
        # flagが0以外の場合、それ以降の処理を省いて次の繰り返しへ。
        if b.flag != 0:
            continue
        # インスタンスbをリストaへ追加。
```

ページ3/6

```
fstar2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
    # インスタンスbをリストaへ追加。
    a.append(b)
    # ファイルを閉じる。
    fp.close()
    # リストaの要素を等級の昇順にソートし、リストbに格納する。
    b = sorted(a, key=lambda a:float(a.mag)) # sort by mag
    # カウンターiを設定。
    i=0
    # リストbの要素を一つずつ取り出し、ブロック内の処理を実施する。
    for c in b:
        # 一つ目の要素(=最も明るい)の等級との差を計算する。
        diff=c.mag - b[0].mag
        # 等級差が上限値未満で、かつ天体数が上限値未満
        # であれば、リストcに追加し、カウンターiを1増やす。
        if diff < diffmag_limit and i < nmax:
            out.append(c.all)
            i+=1
    # 上書きフラグが偽の場合、出力カタログを別名で保持する。
    if owflag == False:
        # バックアップ用のファイル名を作成する: 元ファイル + 「.org」
        catn2 = catn + ".org"
        # ファイル「catn2」がすでに存在していたら、os.remove()で消去する。
        if os.path.isfile(catn2):
            os.remove(catn2)
        # シェルコマンド「mv」のコマンド行を作成する。
        com="mv {0} {1}".format(catn,catn2)
        # subprocess.callメソッドで「mv」を実行する。
        subprocess.call(com,shell=True)
-U:--- fstar2_c.py 52% L134 (Python)---
```

ページ4/6

天体検出ソフトの改良例(3)

~/pylec/samples/pyscripts/fstar2(_c).py (*_c.py:コメント付)

```
fstar2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

subprocess.call(com,shell=True)

# 書き込みモードでファイルを開く。
fp=open(catn,'w')

# リストoutの内容をファイルに書き込む。
fp.writelines(out)

# ファイルを閉じる。
fp.close()

# クラス「scat」の定義:
class scat:

# クラスの初期化時に実行される特殊な関数(__init__)の定義:
def __init__(self,line):
    list=line.split() # 受け取った文字列を空白文字で分割し、リストに格納。
    self.all=line # 受け取った文字列そのものを属
    self.x=float(list[0]) # リストの要素0(中心位置座標x)を属性xに格納。
    self.y=float(list[1]) # リストの要素1(中心位置座標x)を属性yに格納。
    self.n=int(list[2]) # リストの要素2(id番号)を属性nに格納。
    self.fwhm=float(list[3]) # リストの要素3(半値幅)を属性fwhmに格納。
    self.flux=float(list[4]) # リストの要素4(flux)を属性fluxに格納。
    self.ferr=float(list[5]) # リストの要素5(fluxの誤差)を属性ferrに格納。
    self.mag=float(list[6]) # リストの要素6(器械等級)を属性magに格納。
    self.merr=float(list[7]) # リストの要素7(等級の誤差)を属性merrに格納。
    self.fmax=float(list[8]) # リストの要素8(ピーク値)を属性fmaxに格納。
    self.bkg=float(list[9]) # リストの要素9(スカイ値)を属性bkgに格納。
    self.thresh=float(list[10]) # リストの要素10(しきい値)を属性fwhmに格納。
    self.flag=int(list[11]) # リストの要素11(フラッグ)を属性flagに格納。

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
-U:--- fstar2_c.py 69% L166 (Python)-----
```

ページ5/6

```
fstar2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

self.flag=int(list[11]) # リストの要素11(フラッグ)を属性flagに格納。

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

# 用法の表示。引数の数が1以外では用法を出力し、スクリプトを終了する。
# 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
if len(sys.argv) != 2:
    print "Usage: {0} [input-list]".format(sys.argv[0])
    print "cf: {0} object.n.list".format(sys.argv[0])
    sys.exit()

# スクリプト実行時の2個の引数を与え、関数「findstarall」を実行する。
# 「*sys.argv[0:2]」の詳細についてはmkdark_c.pyのコメントを参照。
findstarall(*sys.argv[0:2])

-U:--- fstar2_c.py Bot L180 (Python)-----
```

ページ6/6

練習B4: 画像マッチングソフト(1)

天体画像フレームは、望遠鏡の微小なガイドエラーの影響で、フレーム間で天体の結像位置がずれている。

練習B3で作成した天体検出ソフトの結果を元に、そのずれを補正する画像マッチングソフト「`fimatch1.py`」を作る。

使用タスク: `xyxymatch`, `geomap`, `geotran`

実行: `fimatch1.py` 入力カタログリスト 参照カタログ

実行例: `fimatch1.py object.n.cat.list S0009400.n.cat`

(補足: 参照カタログは便宜上1枚目のフレームを選択。本来は最も多くの天体を好条件で検出したフレームを選択すべき)

練習B4: 画像マッチングソフト(2)

サンプルスクリプトでは、使用タスク一つにつき、一つの関数を作成。

geomap、geotranの関数では、タスク実行前に出力ファイルと同名のファイルが存在していれば削除する。さらに geotranの関数では、タスク実行前にgeomapの出力ファイルの存在とファイルサイズを確認し、存在していない、または空ファイルであれば処理をスキップして次のフレームの処理を進める。

ファイルサイズの確認: `os.path`モジュールの`getsize()`を使用する。

返り値: ファイルサイズのバイト数。

例: `os.path.getsize('ファイル名')`

画像マッチングタスク

xyxymatch: 入力座標リストとリファレンス座標リストとのマッチングを計算。

出力はgeomapの入力に利用できる形式。

用法: **xyxymatch input reference output tolerance**

geomap: 空間変換関数 $[f, in = f(ref)]$ を計算する。

入力はxyxymatchの出力ファイル、出力はgeotranの入力database。

用法: **geomap input database xmin xmax ymin ymax**

geotran: 入力フレームに空間変換関数を適用した結果を返す。

用法: **geotran input output database transforms**

xyxymatchの変数:

デフォルトから値を変更する変数:

xyxymatch: input, reference, output, toleranceの4変数。

input = 天体検出ソフトの出力ファイル (*.cat)

reference = コマンドの引数で指定したもの。

output = input + '.xyxymatch' (*.cat.xyxymatch)

tolerance = 変数化。値はデフォルト値(3)を使用。

geomapの変数:

デフォルトから値を変更する変数:

geomap: input, database, xmin, xmax, ymin, ymax, fitgeometry, interac
の8変数。

input = xyxymatchのoutput (*.cat.xyxymatch)

database = input - '.xyxymatch' + '.geomap' (*.cat.geomap)

xmin = ymin = 1

xmax = ymax = 2048

fitgeometry='shift' (xとyのshiftのみfit)

interac=no

geotranの変数:

デフォルトから値を変更する変数:

geotran: input, output, database, transforms, geometry, nxblock, nyblock, verbose
の8変数。

input = 変換したいフレーム名 (S*.n.fits)

output = 変換後のフレーム名 (S*.s.fits)

database = geomapのdatabase (*.cat.geomap)

transforms = geomapのinput = xyxymatchのoutput (*.cat.xyxymatch)

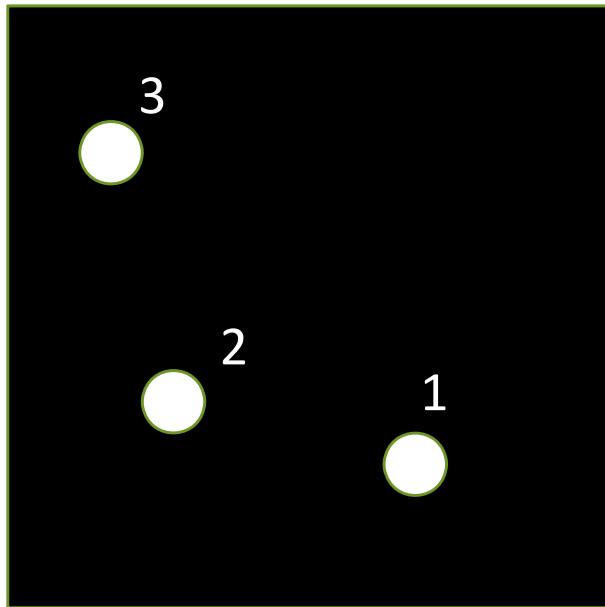
geometry = 'linear' # 空間変換の線形部分のみ適用

nxblock = nyblock = 2048

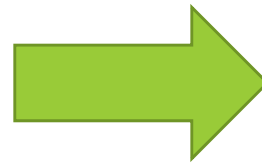
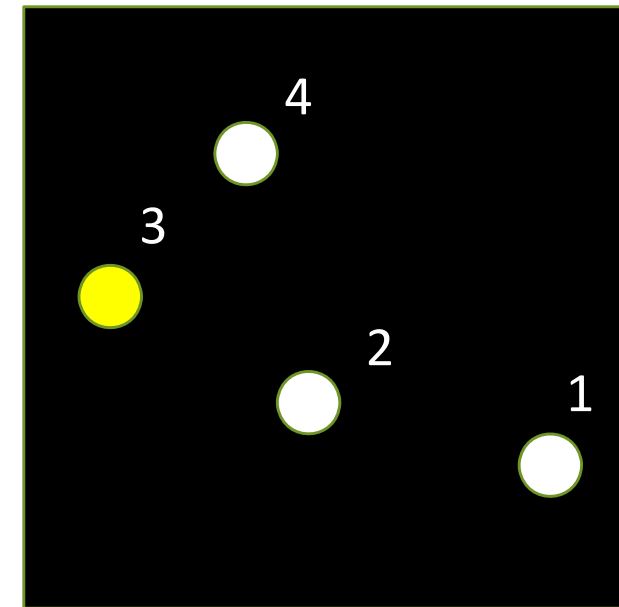
verbose = no

補足: マッチングについて

1枚目のフレーム



XX枚目のフレーム



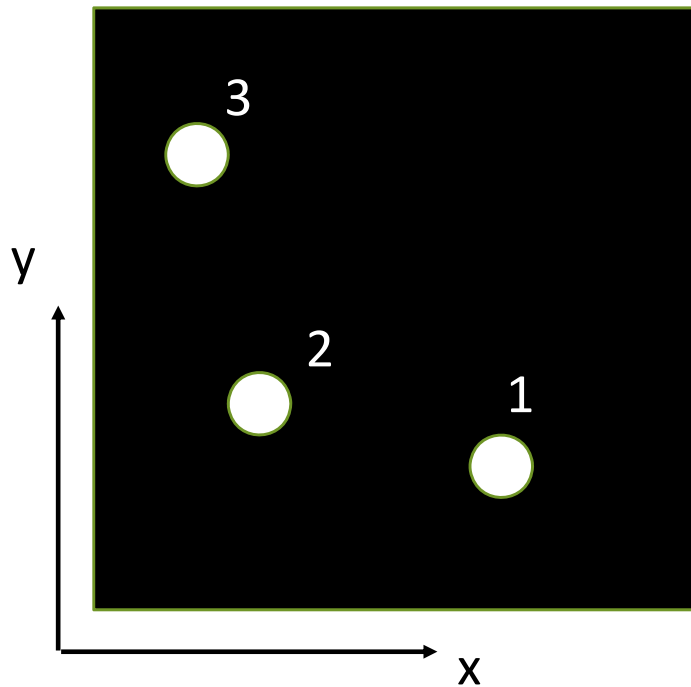
望遠鏡のガイドエラーなどで視野がずれる
= 星の位置がずれる。



星の位置を合わせる
= 本講習での「マッチング」

xyxymatch: 同一星の座標リストを作成

1枚目のフレーム

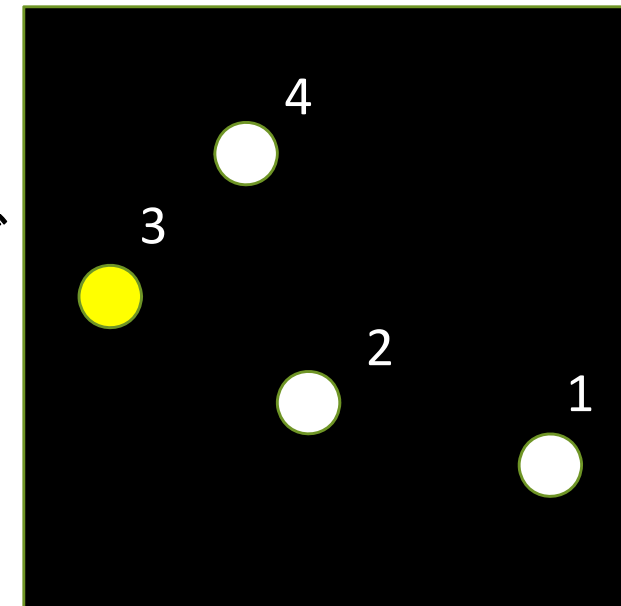


n	x	y
1	700	200
2	300	250
3	200	800

マッピング

n	x	y
1	900	200
2	500	250
3	100	500
4	400	800

XX枚目のフレーム



xyxymatchの結果:
(1枚目、XX枚目)

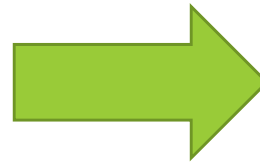
700	200	900	200	1	1
300	250	500	250	2	2
200	800	400	800	3	4

geomap: 画像変換関数の係数を計算

xyxymatchの結果:

700	200	900	200	1	1
300	250	500	250	2	2
200	800	400	800	3	4

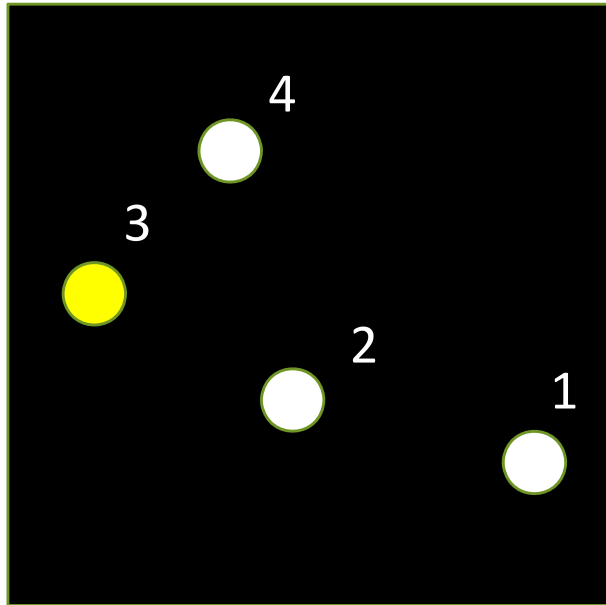
geomap



関数: $f(x,y)$

geotran: 画像変換を実行

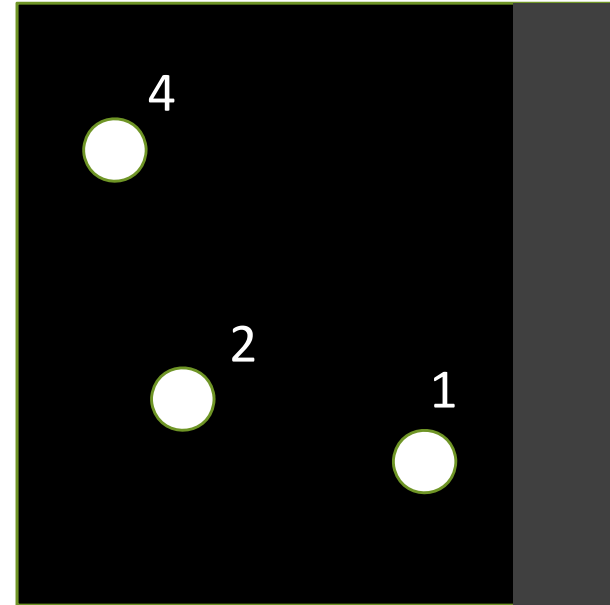
A: XX枚目のフレーム



関数: $f(x,y)$



B: 補正済みフレーム



geotran:

output:B = f (input:A)

補間された
ピクセル。

画像マッチングソフトの例(1)

~/pylec/samples/pyscripts/fimatch1(_c).py (*_c.py:コメント付)

```
fimatch1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
#!/usr/bin/env python
# -- coding: utf-8 --
# 天体検出ソフトの実行結果を元に画像マッチングを実施し、リファレンスフレームとの
# スレを補正したフレームを作成する。
# 処理: 1: 入力リスト内のカタログファイルとリファレンスのカタログとでマッチング。
#       2: マッチング結果を元に、画像変換関数の係数を計算。
#       3: フレームに画像変換を適用し、スレを補正したフレームを作成する。
# M.Isogai

# sysモジュールのimport  引数を使用するため。
import sys
# osモジュールのimport。  ファイルの存在確認 os.path.isfile()、
# ファイルサイズ取得 os.path.getsize()、ファイル削除 os.remove() を使用するため。
import os
# pyraf.irafモジュールのimport。  pythonでirafを使用するため。
from pyraf import iraf

# 関数「fmatch」の定義:
def fmatch(list,ref):

    tol=3.0 # The matching tolerance in pixels in iraf.immatch.xyymatch

# 入力リストを読み込みで開く。
fp=open(list,'r')
# 一行ずつ読み込み、ブロック内でirafのxyymatchタスクを実行する。
for f in fp:
    f=f.strip() # 行頭や行末の空白(改行コード含む)を削除
    # マッチング結果ファイル名(天体カタログ名 + '.xyymatch')を作成。
    g=f + ".xyymatch"

    # xyymatch実行時の引数を表示。動作確認用
    print "fmatch: {0} {1} {2}".format(f,ref,g)

# xyymatchの実行:
-U:--- fimatch1_c.py Top L34 (Python)-----
```

ページ1/5

```
fimatch1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
print "fmatch: {0} {1} {2}".format(f,ref,g)

# xyymatchの実行:
iraf.xyymatch(f,ref,g,tol,separat=9,matchin='triangles',nmatch=30, \
ratio=10,nreject=10,interac='no',verbose='yes')

# ファイルを閉じる。
fp.close()

# 関数「fgeomap」の定義:
def fgeomap(list):

# 入力リストを読み込みで開く。
fp=open(list,'r')
# 一行ずつ読み込み、ブロック内でirafのgeomapタスクを実行する。
for f in fp:
    f=f.strip() # 行頭や行末の空白(改行コード含む)を削除
    # マッチング結果ファイル名(天体カタログ名 + '.xyymatch')を作成。
    g=f + ".xyymatch"
    # geomapの出力ファイル名(天体カタログ名 + '.geomap')を作成。
    h=f + ".geomap"

# geomapの出力ファイル(h)がすでに存在しているか確認(os.path.isfile)し、
# 存在していれば消去(os.remove)する。
if os.path.isfile(h):
    os.remove(h)

# geomap実行時の引数を表示。動作確認用
print "fgeomap: {0} {1}".format(g,h)

# geomapの実行:
iraf.geomap(g,h,xmin=1,xmax=2048,ymin=1,ymax=2048,results='', \
fitgeometry='shift',function='polynomial', \
xxorder=2,xyorder=2,yxorder=2,yyorder=2, \
-U:--- fimatch1_c.py 20% L67 (Python)-----
```

ページ2/5

画像マッチングソフトの例(2)

~/pylec/samples/pyscripts/fimatch1(_c).py (*_c.py:コメント付)

```
fimatch1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

xxorder=2, xyorder=2, yxorder=2, yyorder=2, \
maxiter=0, calctype='real',interac='no')

# ファイルを閉じる。
fp.close()

# 関数「fgeotran」の定義:
def fgeotran(list):

# 入力リストを読み込みで開く。
fp=open(list,'r')
# 一行ずつ読み込み、ブロック内でirafのgeotranタスクを実行する。
for f in fp:
    f=f.strip() # 行頭や行末の空白(改行コード含む)を削除
    # マッチング結果ファイル名(天体カタログ名 + '.xyxymatch')を作成。
    g=f + ".xyxymatch"
    # geomapの出力ファイル名(天体カタログ名 + '.geomap')を作成。
    h=f + ".geomap"

### 入力カタログ名を元に、画像変換前と後のフレーム名を作成する。
# 入力カタログ名の末尾(rstrip)の「.cat」を削除。
i=f.rstrip(".cat")

# カタログ幹名(i)を元に画像変換前と後のフレーム名を作成する。
j=i + ".fits" # 前のフレーム名(*.n.fits)
k=j.replace("n.fits","s.fits") # 後のフレーム名(*.s.fits)

# 画像変換後のフレーム(k)がすでに存在しているか確認(os.path.isfile)し、
# 存在していれば消去(os.remove)する。
if os.path.isfile(k):
    os.remove(k)

# geomapの出力ファイルの存在を確認。もしなければメッセージを表示し、
# これ以降の処理を省略して次の繰り返しの処理に移る。

-U:--- fimatch1_c.py 43% L100 (Python)---
```

ページ3/5

```
fimatch1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

# geomapの出力ファイルの存在を確認。もしなければメッセージを表示し、
# これ以降の処理を省略して次の繰り返しの処理に移る。
if os.path.isfile(h) == False:
    print "!!! Warning !!! {0} is not found! skip...".format(h)
    continue

# geomapの出力ファイルのサイズを確認。もし空ファイル(< 1 Byte)であれば
# メッセージを表示し、これ以降の処理を省略して次の繰り返しの処理に移る。
if os.path.getsize(h) < 1:
    print "!!! Warning !!! {0} is empty! skip...".format(h)
    continue

# geotran実行時の引数を表示。動作確認用
print "geotran {0} {1} {2} {3}".format(j,k,h,g)

# geotranの実行:
iraf.geotran(j,k,h,g,geometry='linear',interpolant='linear', \
            boundary='nearest', constant=0.0, \
            fluxconserve='yes',nxblock=2048, nyblock=2048, \
            verbose='no')

# ファイルを閉じる。
fp.close()

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

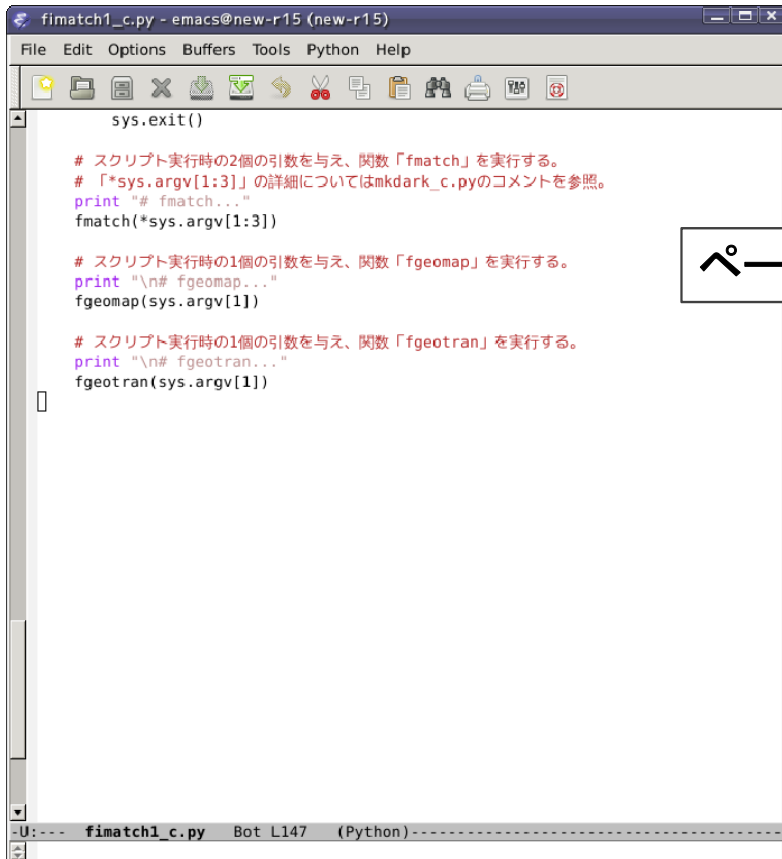
# 用法の表示。引数の数が2以外では用法を出力し、スクリプトを終了する。
# 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
if len(sys.argv) != 3:
    print "Usage: {0} [input-list] [reference-catalog]".format(sys.argv[0])
    print "cf: {0} object.n.cat.list S0009400.n.cat".format(sys.argv[0])
    sys.exit()

-U:--- fimatch1_c.py 65% L134 (Python)---
```

ページ4/5

画像マッチングソフトの例(3)

~/pylec/samples/pyscripts/fimatch1(_c).py (*_c.py:コメント付)



```
fimatch1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
sys.exit()

# スクリプト実行時の2個の引数を与え、関数「fmatch」を実行する。
# 「*sys.argv[1:3]」の詳細についてはmkdark_c.pyのコメントを参照。
print "# fmatch..."
fmatch(*sys.argv[1:3])

# スクリプト実行時の1個の引数を与え、関数「fgeomap」を実行する。
print "\n# fgeomap..."
fgeomap(sys.argv[1])

# スクリプト実行時の1個の引数を与え、関数「fgeotran」を実行する。
print "\n# fgeotran..."
fgeotran(sys.argv[1])

-U:--- fimatch1_c.py Bot L147 (Python)-----
```

ページ5/5

実習B4: 画像マッチングソフトの改良(1)

練習B4の画像マッチングソフトは、subpixelスケールのシフト補正を実施しており、その補正には補間関数を使用している。subpixelスケールの補正は不要なため、補正をx,y方向のpixelスケール(=整数値)のシフトに限定するようソフトを改良する。

改良箇所: geomapの出力ファイルの最後から四行目の二つの数値を
整数値に置換する(数値はxshift, yshift行にも掲載されている)。

→ 詳細は次ページ。

※ 余裕がある受講生向け

geomap出力ファイル

```
S0009718.n.cat.geomap.org - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Org Tbl Help
# Tue 15:30:25 26-Jan-2016
begin S0009718.n.cat.xyymatch
xrefmean 969.2293369140624
yrefmean 940.6413574218749
xmean 1022.733764648438
ymean 901.3599243164062
geometry shift
function polynomial
xshift 53.51345
yshift -39.28137
xmag 1.
ymag 1.
xrotation 0.
yrotation 0.
xrms 0.2015509
yrms 0.2730804
surface1
11
3. 3.
2. 2.
2. 2.
0. 0.
1. 1.
2048. 2048.
1. 1.
2048. 2048.
1. 0.
0. 1.
surface2
0
```

x,yのずれ量(コメント付)

x,yのずれ量(コメントなし)
整数に変換すべき数値

実習B4: 画像マッチングソフトの改良(2)

サンプルスクリプトでは、出力ファイルの係数を整数値化する関数を追加し、geomapを実行する関数にデフォルト引数を追加、このデフォルト引数の値がyesであれば、geomap実行後に整数値化する関数を実行する、としている。

整数値化には、

float() # 文字列 → 実数

round() # 四捨五入

int() # 実数 → 整数

の3つの関数を使用している。

画像マッチングソフトの改良例(1)

~/pylec/samples/pyscripts/fimatch2(_c).py (*_c.py:コメント付)

```

#!/usr/bin/env python
# -- coding: utf-8 --
# 天体検出ソフトの実行結果を元に画像マッチングを実施し、リファレンスフレームとの
# スレを補正したフレームを作成する。
# 処理: 1: 入力リスト内のカタログファイルとリファレンスのカタログとでマッチング。
#       2: マッチング結果を元に、画像変換関数の係数を計算。
#       3: フレームに画像変換を適用し、スレを補正したフレームを作成する。
# ver.2 geomapの出力結果を編集し、xyシフト量を整数値にする処理を追加。
# M.Isogai

# sysモジュールのimport  引数を使用するため。
import sys
# osモジュールのimport。  ファイルの存在確認 os.path.isfile()、
# ファイルサイズ取得 os.path.getsize()、ファイル削除 os.remove() を使用するため。
import os
# pyraf.irafモジュールのimport。  pythonでirafを使用するため。
from pyraf import iraf

# 関数「fmatch」の定義:
def fmatch(list,ref):

    tol=3.0 # The matching tolerance in pixels in iraf.immatch.xyxyatch

    # 入力リストを読み込みで開く。
    fp=open(list,'r')
    # 一行ずつ読み込み、ブロック内でirafのxyxyatchタスクを実行する。
    for f in fp:
        f=f.strip() # 行頭や行末の空白(改行コード含む)を削除
        # マッチング結果ファイル名(天体カタログ名 + '.xyxyatch')を作成。
        gf + ".xyxyatch"

        # xyxyatch実行時の引数を表示。動作確認用
        print "fmatch: {0} {1} {2}".format(f,ref,g)
  
```

ページ1/8

```

print "fmatch: {0} {1} {2}".format(f,ref,g)

# xyxyatchの実行:
iraf.xyxyatch(f,ref,g,tol,separat=9,matchin='triangles',nmatch=30, \
              ratio=10,nreject=10,interac='no',verbose='yes')

# ファイルを閉じる。
fp.close()

# 関数「fgeomap」の定義: 整数化についてのオプション引数useintを追加。
def fgeomap(list,useint=True):

    # 入力リストを読み込みで開く。
    fp=open(list,'r')
    # 一行ずつ読み込み、ブロック内でirafのgeomapタスクを実行する。
    for f in fp:
        f=f.strip() # 行頭や行末の空白(改行コード含む)を削除
        # マッチング結果ファイル名(天体カタログ名 + '.xyxyatch')を作成。
        g=f + ".xyxyatch"
        # geomapの出力ファイル名(天体カタログ名 + '.geomap')を作成。
        h=f + ".geomap"

        # geomapの出力ファイル(h)がすでに存在しているか確認(os.path.isfile)し、
        # 存在していれば消去(os.remove)する。
        if os.path.isfile(h):
            os.remove(h)

        # geomap実行時の引数を表示。動作確認用
        print "fgeomap: {0} {1}".format(g,h)

    # geomapの実行:
    iraf.geomap(g,h,xmin=1,xmax=2048,ymin=1,ymax=2048,results='', \
               fitgeometry='shift', function='polynomial', \
  
```

ページ2/8

画像マッチングソフトの改良例(2)

~/pylec/samples/pyscripts/fimatch2(_c).py (*_c.py:コメント付)

```
fimatch2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
fitgeometry='shift', function='polynomial', \
xxorder=2, xyorder=2, yxorder=2, yyorder=2, \
maxiter=0, calctype='real',interac='no')

# 整数化オプションuseintが真の場合、関数「geomapint」を実行する。
if useint==True:
    geomapint(h)

# ファイルを閉じる。
fp.close()

# 関数「fgeotran」の定義:
def fgeotran(list):

# 入力リストを読み込みで開く。
fp=open(list,'r')
# 一行ずつ読み込み、ブロック内でirafのgeotranタスクを実行する。
for f in fp:
    f=f.strip() # 行頭や行末の空白(改行コード含む)を削除
    # マッチング結果ファイル名(天体カタログ名 + '.xyxymatch')を作成。
    g=f + ".xyxymatch"
    # geomapの出力ファイル名(天体カタログ名 + '.geomap')を作成。
    h=f + ".geomap"

### 入力カタログ名を元に、画像変換前と後のフレーム名を作成する。
# 入力カタログ名の末尾(rstrip)の「.cat」を削除。
i=f.rstrip(".cat")

# カタログ幹名(i)を元に画像変換前と後のフレーム名を作成する。
j=i + ".fits" # 前のフレーム名(*.n.fits)
k=j.replace("n.fits","s.fits") # 後のフレーム名(*.s.fits)

# 画像変換後のフレーム(k)がすでに存在しているか確認(os.path.isfile)し、
# 存在していれば消去(os.remove)する。

-U:--- fimatch2_c.py 28% L102 (Python)-----
```

ページ3/8

```
fimatch2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
# 存在していれば消去(os.remove)する。
if os.path.isfile(k):
    os.remove(k)

# geomapの出力ファイルの存在を確認。もしなければメッセージを表示し、
# これ以降の処理を省略して次の繰り返しに移る。
if os.path.isfile(h) == False:
    print "!!! Warning !!! {0} is not found! skip...".format(h)
    continue

# geomapの出力ファイルのサイズを確認。もし空ファイル(< 1 Byte)であれば
# メッセージを表示し、これ以降の処理を省略して次の繰り返しに移る。
if os.path.getsize(h) < 1:
    print "!!! Warning !!! {0} is empty! skip...".format(h)
    continue

# geotran実行時の引数を表示。動作確認用
print "geotran {0} {1} {2} {3}".format(j,k,h,g)

# geotranの実行:
iraf.geotran(j,k,h,g,geometry='linear',interpolant='linear', \
boundary='nearest', constant=0.0, \
fluxconserve='yes',nxblock=2048, nyblock=2048, \
verbose='no')

# ファイルを閉じる。
fp.close()

# 関数「geomapint」の定義:
def geomapint(inf):
# 入力ファイルを読み込みで開く。
fp=open(inf,'r')

# リスト、変数を初期化:

-U:--- fimatch2_c.py 43% L136 (Python)-----
```

ページ4/8

画像マッチングソフトの改良例(4)

~/pylec/samples/pyscripts/fimatch2(_c).py (*_c.py:コメント付)

```

# x,yのシフト量が両方共文字列に含まれている場合、
elif xsorg in l and ysorg in l:
    # replace()で文字列中のシフト量の数値を整数値に置換する。x,y両方とも。
    m=l.replace(xsorg,str(xsint)).replace(ysorg,str(ysint))
    # 修正した文字列をリストbに追加する。
    b.append(m)
else:
    # 該当しない場合、文字列をそのままリストbに追加。
    b.append(l)

# ファイルを書き込みで開く。
fp=open(inf,'w')
# リストbの内容をファイルに書き込み。
fp.writelines(b)
# ファイルを閉じる。
fp.close()

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

    # 用法の表示。引数の数が2以外では用法を出力し、スクリプトを終了する。
    # 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
    if len(sys.argv) != 3:
        print "Usage: {0} [input-list] [reference-catalog]".format(sys.argv[0])
        print "cf: {0} object.n.cat.list 50009400.n.cat".format(sys.argv[0])
        sys.exit()

    # スクリプト実行時の2個の引数を与え、関数「fmatch」を実行する。
    # 「*sys.argv[1:3]」の詳細についてはmkdark_c.pyのコメントを参照。
    print "# fmatch..."
    fmatch(*sys.argv[1:])

    # スクリプト実行時の1個の引数を与え、関数「fgeomap」を実行する。
    print "\n# fgeomap..."
  
```

ページ7/8

```

print "\n# fgeomap..."
fgeomap(sys.argv[1])

# スクリプト実行時の1個の引数を与え、関数「fgeotran」を実行する。
print "\n# fgeotran..."
fgeotran(sys.argv[1])
  
```

ページ8/8

練習B5: アパーチャ測光ソフト

アパーチャ測光を行うソフト「fphot1.py」を作る。

使用タスク: digiphot.daophot.phot

+ 設定タスク(datapars, fitskypars, centerpars, photpars)

実行: fphot1.py 入力リスト 天体位置座標ファイル PSF半値幅(=fwhm)

実行例(100918): [fphot1.py](#) [object.s.list](#) [Sg.coo](#) 7

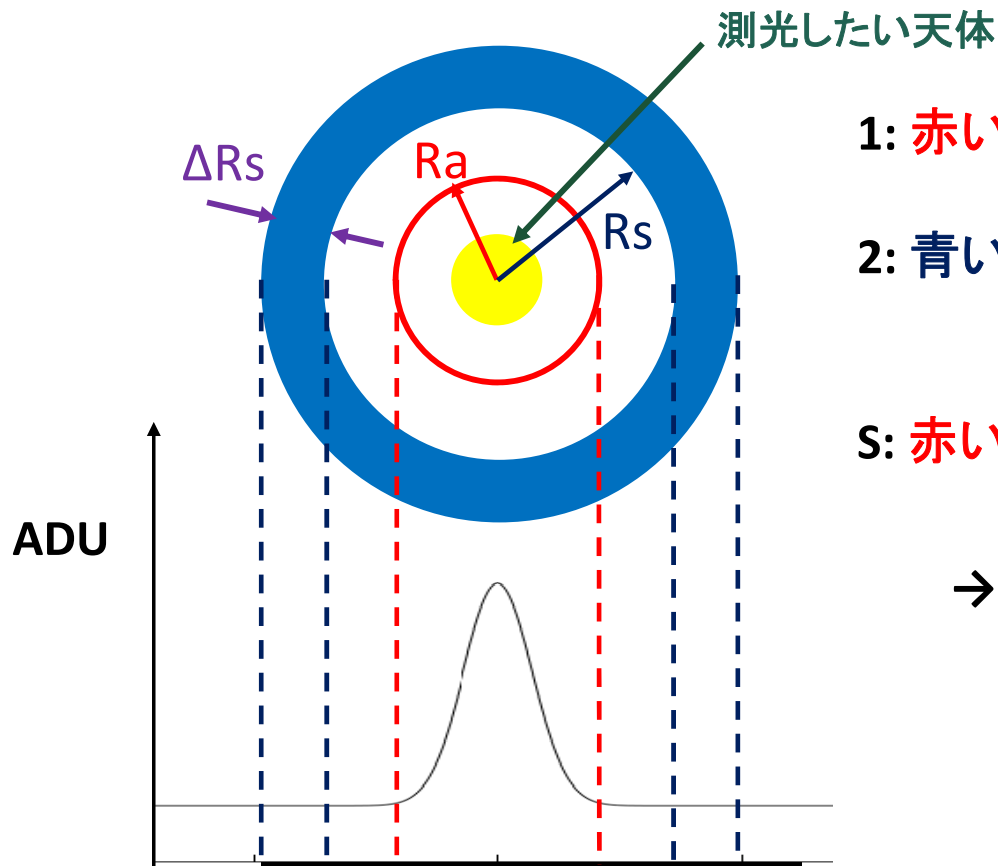
補足:

天体位置座標ファイル: 測光したい天体の座標(x,y)一覧のファイル。

画像マッチングのリファレンスカタログを元に作成。

PSF半値幅: fstar1.pyの出力カタログの半値幅を元に決定。

アパーチャ(=開口)測光



- 1: 赤い円内のADU値の積分値
= 星+スカイのフラックス
 - 2: 青い円環内のADU値のmedian
= スカイのADU値(1pixelあたり)
- S: 赤い円の面積

→ 星の総ADU値 = 1 - 2 * S

- R_a** : アパーチャ半径: photの引数 (photpar)
 R_s : スカイ円環半径: photの引数 (fitskypars)
 ΔR_s : スカイ円環幅: photの引数(fitskypars)

測光タスクについて

photタスクを実行する前に、以下のタスクの隠し引数の設定をおこなう。

- datapars: PSFの半値幅やデータに関する情報を設定
- centerpars: アパーチャの中心位置測定に関する変数を設定
- fitskypars: スカイ値測定に関する変数を設定
- photpars: 測光に関する変数を設定

※ これらのタスクは非対話での実行が不可。隠し引数の設定は、
デフォルト値の変更方法で実施。

各タスクの変数設定値

以下で明示した変数以外はデフォルト値を使用:

- **datapars:** fwhmpsf=引数で指定したfwhm,
sig=skyのsigmaを測定して入力,
datamin=0, datamax=50000, ccdread='ronoise',
gain='gain', exposur='exp-time', filter='filter-1',
obstime='ut'
- **centerpars:** calgori='centroid', cbox=mxs=fwhmの2倍
- **fitskypars:** salgori='mode', annulus=fwhmの5倍、dannulu=10
- **photpars:** apertures = fwhmの2倍

サンプルスクリプトの補足

サンプルスクリプトでは、

- ・ 隠し引数設定タスクはphotpar以外、個々に関数を定義・実行
- ・ 設定後に設定値を表示
- ・ digiphotパッケージとdaophotパッケージのloadを確認し、loadされていなければスクリプト内でload

としている。

さらに、アパーチャ径、スカイ円環内径、円環幅の計算式および値は本講習用に便宜的に採用したもので推奨値ではない。これらの値はそれぞれの観測環境・装置・条件に応じて適宜調整が必要。

アパーチャ測光ソフトの例(1)

~/pilec/samples/pyscripts/fphot1(_c).py (*_c.py:コメント付)

```
#!/usr/bin/env python
# -- coding: utf-8 --
# daophot.photタスクを使用してアパーチャ測光を実施する。
# 測光のための各種パラメータは、全フレーム共通のものに関しては関数内で設定する。
# M.Isogai

# sysモジュールのimport  引数を使用するため。
import sys
# pyraf.irafモジュールのimport。  pythonでirafを使用するため。
from pyraf import iraf

# 関数「fphot」の定義:
def fphot(list, fcoo, fwhm):

    # モジュール名が長くなるので略記号を定義して使用する。
    i = iraf.daophot

    # 共通パラメータの値を用意。ここでは以下の値を採用する。
    dmin=0      # datapars.dataminのデフォルト値
    dmax=50000  # datapars.datamaxのデフォルト値
    danu=10     # fitskypars.dannulusのデフォルト値

    # 引数fwhmのデータ型を実数型に変換する。
    fwhm=float(fwhm)

    # digiphot/パッケージがloadされていなければ、メッセージを表示してloadする。
    if not iraf.defpac('digiphot'):
        print "load digiphot..."
        iraf.digiphot()

    # daophot/パッケージがloadされていなければ、メッセージを表示してloadする。
    if not iraf.defpac('daophot'):
        print "load daophot..."
        iraf.daophot()
```

ページ1/5

```
iraf.daophot()

# setdatapars関数を実行し、dataparsのデフォルト値を設定する。
sig=0.0
setdatapars(fwhm, sig, dmin, dmax)
print "=====" datapars "====="
# 設定値を表示する:
iraf.lparam(i.datapars)

# setcenterpars関数を実行し、centerparsのデフォルト値を設定する。
mxs=cbx=0.0
setcenterpars(cbx, mxs)
print "=====" centerpars "====="
# 設定値を表示する:
iraf.lparam(i.centerpars)

# setfitskypars関数を実行し、fitskyparsのデフォルト値を設定する。
anu=0
setfitskypars(anu, danu)
print "=====" fitskypars "====="
# 設定値を表示する:
iraf.lparam(i.fitskypars)

# photparsのデフォルト値を設定する:
j=i.photpars
j.weighting='constant'
j.zmag=25.
j.mkapert='no'

# 入力リストを読み込みで開く。
fp=open(list, 'r')
# 一行ずつ読み込み、ブロック内でirafのphotタスクを実行する。
for f in fp:
    f = f.strip() # 行頭や行末の空白(改行コード含む)を削除
```

ページ2/5

アパーチャ測光ソフトの例(2)

~/pylec/samples/pyscripts/fphot1(_c).py (*_c.py:コメント付)

```
fphot1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

f = f.strip() # 行頭や行末の空白(改行コード含む)を削除

# irafのimstatでフレームの標準偏差を計算し、結果をsigに格納する。
sig = iraf.imstat(f,format='no',fields='stddev',nclip=2, Stdout=1)[0]
# sigの値を表示する。
print "# ===== {0}: sky-sigma: {1}".format(f,sig)
# sigの値をdataparsのsigmaパラメータの値として設定。
i.datapars.sigma=sig

# 引数fwhmの値を元にcbx, mxsを計算。centerparsのcboxとmaxshifの
# デフォルト値に設定。
cbx = 2.0 * fwhm
mxs = cbx
i.centerpars.cbox=cbx
i.centerpars.maxshif=mxs

# 引数fwhmの値を元にanuを計算。fitskyparsのannulus変数の
# デフォルト値に設定。
anu = 5.0 * fwhm
i.fitskypars.annulus=anu

# 引数fwhmの値を元にapeを計算。fitskyparsのapertures変数の
# デフォルト値に設定。
ape = 2.0 * fwhm
j.apertures=ape

# photを実行。
i.phot(f,fcoo,'default','', plotfil='', datapar='', \
      centerp='', fitskyp='', photpar='', \
      interactive='no', radplot='no', \
      cache='yes', verify='no')

# ファイルを閉じる。
fp.close()
```

ページ3/5

```
fphot1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

fp.close()

# 関数「setdatapars」の定義: dataparsのパラメータのデフォルト値を設定する。
def setdatapars(fwhm,sig,dmin,dmax):
    i=iraf.daophot.datapars
    i.scale=1
    i.fwhmpsf=fwhm
    i.sigma=sig
    i.datamin=dmin
    i.datamax=dmax
    i.ccdread='ronoise'
    i.gain='gain'
    i.exposur='exp-time'
    i.filter='filter-1'
    i.obstime='ut'

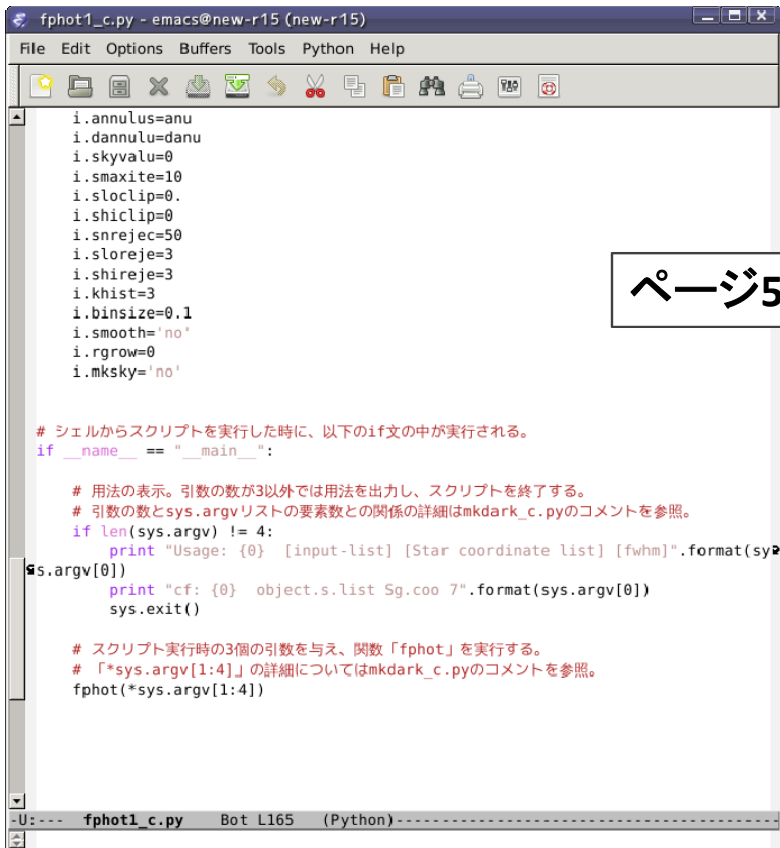
# 関数「setcenterpars」の定義: centerparsのパラメータのデフォルト値を設定する。
def setcenterpars(cbx,mxs):
    i = iraf.daophot.centerpars
    i.categori='centroid'
    i.cbox=cbx
    i.cthresh=0.0
    i.minsnra=1
    i.cmaxite=10
    i.maxshif=mxs
    i.clean='no'

# 関数「setfitskypars」の定義: fitskyparsのパラメータのデフォルト値を設定する。
def setfitskypars(anu,danu):
    i = iraf.daophot.fitskypars
    i.sategori='mode'
    i.annulus=anu
    i.dannulu=danu
```

ページ4/5

アパーチャ測光ソフトの例(3)

~/pilec/samples/pyscripts/fphot1(_c).py (*_c.py:コメント付)



```
fphot1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help
i.annulus=anu
i.dannulu=danu
i.skyvalu=0
i.smaxite=10
i.sloclip=0.
i.shiclip=0
i.snrejec=50
i.sloreje=3
i.shireje=3
i.khist=3
i.binsize=0.1
i.smooth='no'
i.rgrow=0
i.mksky='no'

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

    # 用法の表示。引数の数が3以外では用法を出力し、スクリプトを終了する。
    # 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
    if len(sys.argv) != 4:
        print "Usage: {0} [input-list] [Star coordinate list] [fwhm]".format(sys
sys.argv[0])
        print "cf: {0} object.s.list Sg.coo 7".format(sys.argv[0])
        sys.exit()

    # スクリプト実行時の3個の引数を与え、関数「fphot」を実行する。
    # 「*sys.argv[1:4]」の詳細についてはmkdark_c.pyのコメントを参照。
    fphot(*sys.argv[1:4])

-U:--- fphot1_c.py Bot L165 (Python)-----
```

ページ5/5

位置座標ファイルの作成と目的天体

リファレンスフレームと、そのカタログを
ds9に表示する:

--> `display S0009400.s.fits 1`

--> `tvmark 1 S0009400.n.cat mark=circle`

`radii='25,26,27' color=205 tysize=10`

`label+ inter-` (※1行で書く)

目的天体: 7番

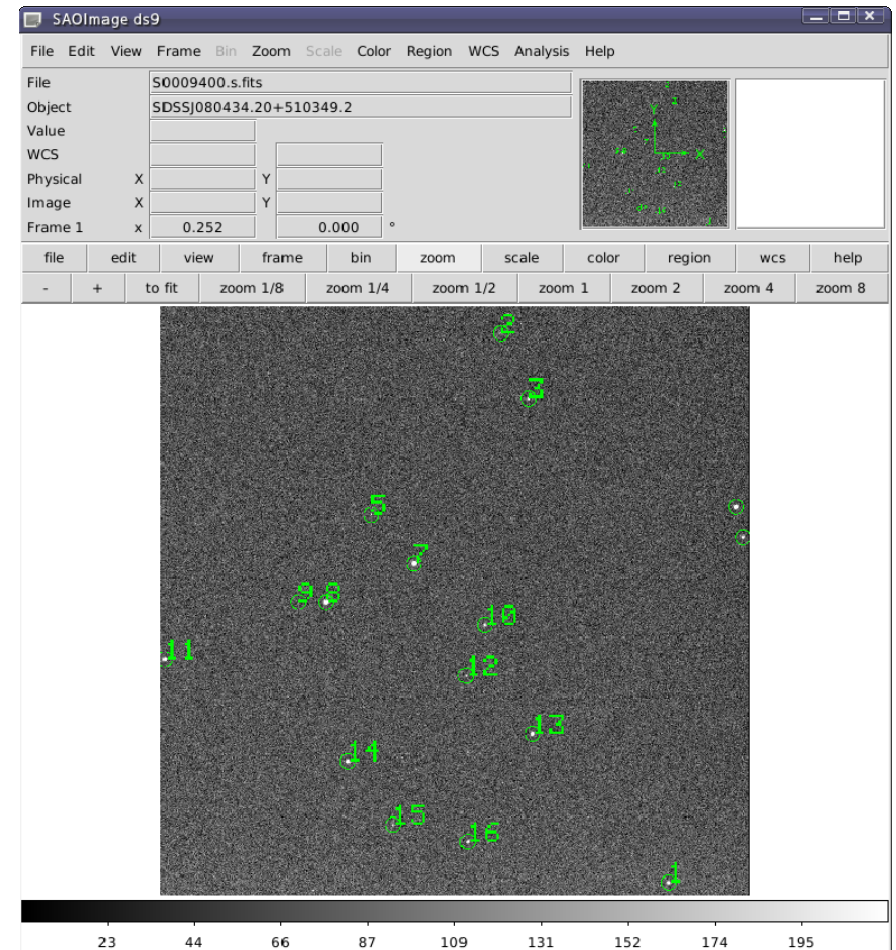
比較星: 8番 (+ 10番, 13番, 14番)

を使用する。

→ 位置座標ファイル:

一行で一つの星: `x y ...`

(例: `100918/Sg.coo`)



練習B6: 比較測光ソフト(1)

測光結果ファイルを元に、比較星とターゲット天体との等級差を計算し、出力するソフトを作る。ソフトにはヘッダ情報のユリウス日(JD)を取得し、修正ユリウス日(MJD \equiv JD - 2,400,000.5)と1枚目のフレームからの経過日(=JD_{xx} - JD₁)を計算して出力する機能を含める。タスク実行前に、digiphotおよびptoolsパッケージのloadを確認し、loadされていない場合はloadする。

使用タスク: pdump (=digiphot.ptools.pdump)

実行: fpdump1.py 入力リスト ターゲット天体番号 比較星番号 ¥
出力ファイル名

pdumpの用法: **入力ファイル フィールド名 実行条件 ¥**
隠し引数(header='no' parameter='yes')

pdumpの用法(詳細版)

pdumpの用法: **入力ファイル フィールド名 実行条件 header='no' parameter='yes'**

- ・ 入力ファイル: photタスクの出力ファイル
- ・ フィールド名: 出力したいフィールド名 (=情報名)を指定。
ここでは「image, id, xcen, ycen, mag,merr, cier, sier, pier」を指定。
(image, id: 入力ファイル名と星のid番号、
xcen,ycen:中心座標、
mag,merr: 器械等級と誤差、
cier, sier, pier: 各処理の実行結果情報)
- ・ 実行条件: 実行を制限する条件式を指定。
ここでは、id == ターゲット星番号 と id == 比較星番号。
(= それぞれターゲット星の結果、比較星の結果のみ出力)

fitsヘッダ情報の取得

fitsヘッダ情報の取得:

- astropy.ioモジュールの「fits」を使用:

用法:

```
from astropy.io import fits
```

```
hl = fits.open('fitsファイル名')
```

```
jd = float( hl[0].header['jd'])
```

```
# hl[0]はPrimary HDU。 (HDU: Header Data Unit)
```

```
# ここからキーワード「jd」の値を取得し、float()で実数に変換。
```

サンプルスクリプトの補足

サンプルスクリプトについての補足。

- ・各フレームについて、目的天体、比較星と二回pdumpを実行している。
- ・測光結果の格納にはstar型のクラスを定義・使用している。等級差とその誤差の計算には、star型クラスのインスタンスを作成し、その属性を用いているが、**クラスを使わなくても実現可能である。**

(クラスを使う方を推奨: スクリプトの可読性が向上するため)

比較測光ソフトの例(1)

~/pylec/samples/pyscripts/fpdump1(_c).py (*_c.py:コメント付)

```

fpdump1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

#!/usr/bin/env python
# -- coding: utf-8 --
# ptools.pdumpタスクを使用して測光結果を受け取り、比較星との等級差を計算する。
# M.Isogai

# sysモジュールのimport  引数を使用するため。
import sys
# pyraf.irafモジュールのimport。  pythonでirafを使用するため。
from pyraf import iraf
# astropy.io.fitsモジュールのimport。 fitsヘッダの値入手で使用するため。
from astropy.io import fits

# 関数「fpdump」の定義:
def fpdump(list,id_obj,id_cmp,ofn):

    # 出力ファイルを開く。
    fp2=open(ofn,'w')

    # pdumpの引数fieldsを指定。
    # 測光結果の出力のうち、フレーム名、id番号、中心位置(x,y)、器械等級、
    # その誤差、各処理(再中心化、スカイ差し引き、測光)のステータス
    # (cier, sier, pier)を入手する。詳細は「phelp phot」を参照。
    fields='image,id,xcen,ycen,mag,merr,cier,sier,pier'

    # digiphotパッケージがloadされていなければ、メッセージを表示してloadする。
    if not iraf.defpac('digiphot'):
        print "# load digiphot..."
        iraf.digiphot()

    # ptoolsパッケージがloadされていなければ、メッセージを表示してloadする。
    if not iraf.defpac('ptools'):
        print "# load ptools..."
        iraf.ptools()

-U:--- fpdump1_c.py  Top L34  (Python)-----

```

ページ1/4

```

fpdump1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

iraf.ptools()

# 入力リストを読み込みで開く。
fp=open(list,'r')
# 初期値を設定しておく。
jd0=0.0
i=0
# 一行ずつ読み込み、ブロック内でirafのphotタスクを実行する。
for f in fp:
    f = f.strip() # 行頭や行末の空白(改行コード含む)を削除

    # pdumpを実行し、ターゲット天体の結果を変数aに格納する。
    # 実行結果はリスト(の1番目の要素)で返るため、1番目の要素のみを
    # 受け取るため[0]をつける。
    a=iraf.pdump(f,fields,'id='+ str(id_obj), header='no', \
        paramet='yes', Stdout=1)[0]

    # pdumpを実行し、比較星の結果を変数bに格納する。書式についてはaと同じ。
    b=iraf.pdump(f,fields,'id='+ str(id_cmp), header='no', \
        paramet='yes', Stdout=1)[0]

    # ターゲット天体の結果を元に、クラスstarのインスタンスsoを作成する。
    so = star(a)
    # 比較星の結果を元に、クラスstarのインスタンスscを作成する。
    sc = star(b)

    # ターゲット天体と比較星との等級差とその誤差を計算する。
    diffm = so.m - sc.m
    err = ( so.me**2 + sc.me**2 )**0.5
    # フレーム名を変数fnに格納する。
    fn = so.imn

    # fitsヘッダーのキーワードjd(ユリウス日)の値を取得し、修正ユリウス日を
    # 計算する。
    hl = fits.open(fn)

-U:--- fpdump1_c.py  23% L68  (Python)-----

```

ページ2/4

比較測光ソフトの例(2)

~/pylec/samples/pyscripts/fpdump1(_c).py (*_c.py:コメント付)

```

fpdump1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

hl = fits.open(fn)
jd = float( hl[0].header['jd'] )
mjd = jd - 2400000.5

# カウンターiが0の場合、ユリウス日を別変数に保持しておく。
# 経過時間も出力に追加するため。
if i == 0:
    jd0 = jd

# 1枚目のフレームからの経過時間(単位:日)を計算する
djd = jd - jd0

# 出力情報を整形して変数に格納する:
output="{0} {13:11.5f} {14:7.5f} {1:6.3f} {2:5.3f} {3:6.3f} {4:5.3f}
{5:6.3f} {6:5.3f} {7} {8} {9} {10} {11} {12}".format(fn, diffm, err, so.m, s
fo.me, sc.m, sc.me, so.cier, so.sier, so.pier, sc.cier, sc.sier, sc.pier, mjd, dj
sd)

# 結果を標準出力に表示する:
print output

# 結果をファイル出力する場合、結果をファイルへも出力する。
fp2.write(output + '\n')

# カウンターiを1増やす。
i+=1

# ファイルを閉じる。
fp.close()

# 標準出力にメッセージを表示し、出力ファイルを閉じる:
print " ==> make {0}".format(ofn)
fp2.close()

# クラス「star」の定義:
class star:

```

ページ3/4

```

fpdump1_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

# クラス「star」の定義:
class star:

# クラスの初期化時に実行される特殊な関数(__init__)の定義:
def __init__(self, line):
    list = line.split() # 受け取った文字列を空白文字で分割し、リストに格納。
    self.all = line # 受け取った文字列そのものを属性allに格納。
    self.imn = list[0] # リストの要素0(フレーム名)を属性imnに格納。
    self.id = int(list[1]) # リストの要素1(id番号)を属性imnに格納。
    self.x = float(list[2]) # リストの要素2(中心位置座標x)を属性xに格納。
    self.y = float(list[3]) # リストの要素3(中心位置座標y)を属性yに格納。
    self.m = float(list[4]) # リストの要素4(等級m)を属性mに格納。
    self.me = float(list[5]) # リストの要素5(等級の誤差me)を属性meに格納。
    self.cier = int(list[6]) # リストの要素6(cier)を属性cierに格納。
    self.sier = int(list[7]) # リストの要素7(sier)を属性sierに格納。
    self.pier = int(list[8]) # リストの要素8(pier)を属性pierに格納。

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

# 用法の表示。引数の数が3または4以外では用法を出力し、スクリプトを終了する。
# 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
if len(sys.argv) != 5:
    print "Usage: {0} [input-list] [id_object] [id_comparison] [output file]
{name} ".format(sys.argv[0])
    print "cf: {0} object.s.mag1.list 1 2 outphot.dat ".format(sys.argv[0]
s])

    sys.exit()

# スクリプト実行時の3個の引数を与え、関数「fpdump」を実行する。
# 「*sys.argv[1:]」は2つ目以降の全ての要素を展開する。
fpdump(*sys.argv[1:])

```

ページ4/4

実習B6: 比較測光ソフトの改良(1)

pythonのグラフプロットライブラリmatplotlibのpyplotサブモジュールを用いて、計算した等級差をグラフ化するオプションを追加する。

コマンド実行時に「-pl」を追加すると、グラフを表示するように改良する。

さらに、出力ファイル名をオプション引数化する。指定が無ければファイルを出力しないようにする。

※ 余裕がある受講生向け

実習B6: 比較測光ソフトの改良(2)

1. matplotlib.pyplotの用法: (x, y, yerr それぞれのリストが必要)

```
import matplotlib.pyplot as plt    # matplotlib.pyplotをimportし、pltとする
```

```
plt.plot(xdata, ydata, 'b.', label='ラベル名')
```

データ点のplot

(xdata, ydata: データ点のx,yのリスト、

b.: データ点の色と形状:「.」でblue)

```
plt.errorbar(xdata, ydata, yerr=ydataerr, fmt='none', capsize=0)
```

誤差棒のplot (ydataerr: 誤差のリスト)

```
plt.gca().invert_yaxis()    # y軸を逆向きに (数値:小 が上)
```

```
plt.show()    # グラフを表示
```

実習B6: 比較測光ソフトの改良(3)

2. グラフ描写のオプション化:

グラフ描写の関数を追加し、比較測光を実施する関数にデフォルト引数「plot=False」を追加。この引数がTrueの場合に、グラフ描写関数を実行する。

3. ファイル出力のオプション化:

比較測光を実施する関数の引数の最後に「*args」(=余った引数を全て受け取るタプル)を追加。タプル「args」の要素数が0より大きければ、最初の要素を出力ファイル名として受け取り、このファイルへ出力する。

比較測光ソフトの改良例(1)

~/pylec/samples/pyscripts/fpdump2(_c).py (*_c.py:コメント付)

```

#!/usr/bin/env python
# -- coding: utf-8 --
# ptools.pdumpタスクを使用して測光結果を受け取り、比較星との等級差を計算する。
# ver.2: matplotlib.pyplotによるグラフ表示を追加。
# M.Isogai

# sysモジュールのimport 引数を使用するため。
import sys
# pyraf.irafモジュールのimport。 pythonでirafを使用するため。
from pyraf import iraf
# astropy.io.fitsモジュールのimport。 fitsヘッダの値入手で使用するため。
from astropy.io import fits
# matplotlib.pyplotをpltという名前でもimport。比較測光結果のグラフ表示のため。
import matplotlib.pyplot as plt

# 関数「fpdump」の定義: 引数にグラフ表示するかどうかのオプション引数plotを追加。
def fpdump(list,id_obj,id_cmp, plot=False, *args):

    # plotするかどうかを表示する。動作確認用。
    print "plot: [" + str(flag_plot) + "]"
    plot=bool(plot)

    # 引数で出力ファイル名を指定していた場合、結果をそのファイルへ出力する。
    flag_ofn=False
    if len(args) > 0:
        flag_ofn=True
        ofn=args[0]
        fp2=open(ofn,'w')

    # pdumpの引数fieldsを指定。
    # 測光結果の出力のうち、フレーム名、id番号、中心位置(x,y)、器械等級、
    # その誤差、各処理(再中心化、スカイ差し引き、測光)のステータス
    # (cier, sier, pier)を入手する。詳細は「pHELP phot」を参照。
    fields='image,id,xcen,ycen,mag,merr,cier,sier,pier'

```

ページ1/6

```

fields='image,id,xcen,ycen,mag,merr,cier,sier,pier'

# digiphotパッケージがloadされていない場合は、メッセージを表示してloadする。
if not iraf.defpac('digiphot'):
    print "# load digiphot..."
    iraf.digiphot()

# ptoolsパッケージがloadされていない場合は、メッセージを表示してloadする。
if not iraf.defpac('ptools'):
    print "# load ptools..."
    iraf.ptools()

# 入力リストを読み込みで開く。
fp=open(list,'r')
# 初期値を設定しておく。
jd0=0.0
i=0
x,y,ye=[],[],[] # list for plot
# 一行ずつ読み込み、ブロック内でirafのphotタスクを実行する。
for f in fp:
    f = f.strip() # 行頭や行末の空白(改行コード含む)を削除

    # pdumpを実行し、ターゲット天体の結果を変数aに格納する。
    # 実行結果はリスト(の1番目の要素)で返るため、1番目の要素のみを
    # 受け取るため[0]をつける。
    a=iraf.pdump(f,fields,'id='+ str(id_obj), header='no', \
        paramet='yes', Stdout=1)[0]

    # pdumpを実行し、比較星の結果を変数bに格納する。書式についてはaと同じ。
    b=iraf.pdump(f,fields,'id='+ str(id_cmp), header='no', \
        paramet='yes', Stdout=1)[0]

    # ターゲット天体の結果を元に、クラスstarのインスタンスsoを作成する。
    so = star(a)
    # 比較星の結果を元に、クラスstarのインスタンスscを作成する。

```

ページ2/6

比較測光ソフトの改良例(2)

~/pylec/samples/pyscripts/fpdump2(_c).py (*_c.py:コメント付)

```

fpdump2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

# 比較星の結果を元に、クラスstarのインスタンスscを作成する。
sc = star(b)

# ターゲット天体と比較星との等級差とその誤差を計算する。
diffm = so.m - sc.m
err = ( so.me**2 + sc.me**2 )**0.5
# フレーム名を変数fnに格納する。
fn = so.imn

# fitsヘッダーのキーワードjd(ユリウス日)の値を取得し、修正ユリウス日を
# 計算する。
hl = fits.open(fn)
jd = float( hl[0].header['jd'] )
mjd = jd - 2400000.5

# カウンターiが0の場合、ユリウス日を別変数に保持しておく。
# 経過時間も出力に追加するため。
if i == 0:
    jd0 = jd

# 1枚目のフレームからの経過時間(単位:日)を計算する
djd = jd - jd0

# 出力情報を整形して変数に格納する:
output="{0} {13:11.5f} {14:7.5f} {1:6.3f} {2:5.3f} {3:6.3f} {4:5.3f}
{5:6.3f} {6:5.3f} {7} {8} {9} {10} {11} {12}".format(fn, diffm, err, so.m, so
so.me, sc.m, sc.me, so.cier, so.sier, so.pier, sc.cier, sc.sier, sc.pier, mjd, dj
sd)

# 結果を標準出力に表示する:
print output

# plot用のリストx,y,yeにそれぞれdjd, diffm, errの値を追加する。
x.append(djd)
y.append(diffm)
ye.append(err)

```

ページ3/6

```

fpdump2_c.py - emacs@new-r15 (new-r15)
File Edit Options Buffers Tools Python Help

ye.append(err)

# 結果をファイル出力する場合、結果をファイルへも出力する。
if flag_ofn == True:
    fp2.write(output + '\n')

# カウンターiを1増やす。
i+=1

# ファイルを閉じる。
fp.close()

# 結果をファイル出力する場合、標準出力にメッセージを表示し、出力ファイルも
# 閉じる:
if flag_ofn == True:
    print " ==> make {0}".format(ofn)
    fp2.close()

# オプション引数plotが真の場合、メッセージを表示して関数「plotlc」を実行する。
if plot == True:
    print " ==> plot light-curve"
    plotlc(x,y,ye)

# クラス「star」の定義:
class star:

# クラスの初期化時に実行される特殊な関数(__init__)の定義:
def __init__(self,line):
    list = line.split() # 受け取った文字列を空白文字で分割し、リストに格納。
    self.all = line # 受け取った文字列そのものを属性allに格納。
    self.imn = list[0] # リストの要素0(フレーム名)を属性imnに格納。
    self.id = int(list[1]) # リストの要素1(id番号)を属性idに格納。
    self.x = float(list[2]) # リストの要素2(中心位置座標x)を属性xに格納。

```

ページ4/6

比較測光ソフトの改良例(3)

~/pylec/samples/pyscripts/fpdump2(_c).py (*_c.py:コメント付)

```

self.x = float(list[2]) # リストの要素2(中心位置座標x)を属性xに格納。
self.y = float(list[3]) # リストの要素3(中心位置座標y)を属性yに格納。
self.m = float(list[4]) # リストの要素4(等級m)を属性mに格納。
self.me = float(list[5]) # リストの要素5(等級の誤差me)を属性meに格納。
self.cier = int(list[6]) # リストの要素6(cier)を属性cierに格納。
self.sier = int(list[7]) # リストの要素7(sier)を属性sierに格納。
self.pier = int(list[8]) # リストの要素8(pier)を属性pierに格納。

# 関数「plotlc」の定義:
def plotlc(x,y,ye):

    # 青点でデータ点をプロットする。ラベルは「diff mag」
    plt.plot(x,y,'b.', label='diff mag')
    # cap(横棒)なしで誤差棒の表示(のみ)を追加する。
    plt.errorbar(x,y,yerr=ye,fmt='none',capsize=0)

    plt.legend() # 凡例の表示
    plt.xlabel('dMJD') # x軸ラベルの設定
    plt.ylabel('diff mag (obj - Comp)') # y軸ラベルの設定
    plt.title('Light Curve') # グラフタイトルの設定
    plt.gca().invert_yaxis() # y軸の向きを変更(数字が小さい方が上に)
    plt.show() # グラフを表示する。

# シェルからスクリプトを実行した時に、以下のif文の中が実行される。
if __name__ == "__main__":

    # 用法の表示。引数の数が3,4,5以外では用法を出力し、スクリプトを終了する。
    # 引数の数とsys.argvリストの要素数との関係の詳細はmkdark_c.pyのコメントを参照。
    if len(sys.argv) < 4 or len(sys.argv) > 6:
        print "Usage: {0} (-pl) [input-list] [id_object] [id_comparison] [output-
filename)".format(sys.argv[0])
        print "cf: {0} (-pl) object.s.mag1.list 1 2 outphot.dat".format(sys.
argv[0])
    else:
        fpdump(*sys.argv[1:])

```

ページ5/6

```

print "cf: {0} (-pl) object.s.mag1.list 1 2 outphot.dat".format(sys.
argv[0])
print " -pl: plot light curve"
sys.exit()

# 引数の判定。もし引数として「-pl」があれば、引数リストからこの要素を除外し、
# 変数 flag_plot の値を真とする。
flag_plot=False
for n in sys.argv:
    if '-pl' in n:
        sys.argv.remove(n)
        flag_plot=True
        break

# sys.argvリストの5番目の要素にflag_plotの値を挿入。
sys.argv.insert(4,flag_plot)

# スクリプト実行時の3個の引数を与え、関数「fpdump」を実行する。
# 「*sys.argv[1:]」は2つ目以降の全ての要素を展開する。
fpdump(*sys.argv[1:])

```

ページ6/6

実習B7: 100925のデータ処理

100918の比較測光まで終了し、さらに時間に余裕があれば100925のデータも処理する。

リファレンスフレームと位置座標ファイルは100918と同じものを使用する。

アパーチャ測光を行う際に指定する半値幅は、カタログファイルの半値幅を参照して値を変更する。

※ 余裕がある受講生向け

終わりに(1): サンプルスクリプトの補足

以上でPythonスクリプト実習は終了です。

この実習で、PythonでPyRAFを使用したスクリプトの書き方を身に付けることができたと思います。

なお、実習のサンプルスクリプトは、**例外処理・エラー処理を意図的に省いています**。

実用に耐える解析ソフトには、これらの処理が必要ですので注意してください。

終わりに(2):本格的な解析ソフトへ

さらに、本実習で作成したソフトは、時間の制約上、**必要最小限の機能**しか実装していません。

一次処理と画像マッチングを自動で行うなど、本格的な解析ソフトへの改良には、観測ログの自動作成、各種フレームリストの自動作成、リファレンスフレームの自動選定、天体非検出フレームやマッチング失敗フレームの処理、などの機能の実装が必要ですがこれらは全て実習で作成したソフトを元にPythonで実現可能です。興味・必要があれば挑戦してみましょう。

PyRAF+Python参考文献

テキスト作成にあたり、以下を参考にしている。

- PyRAF Tutorial

http://stsdas.stsci.edu/pyraf/doc.old/pyraf_tutorial/

- PyRAF Programmer's guide

http://stsdas.stsci.edu/pyraf/doc.old/pyraf_guide/pyraf_guide.html

- 過去のADC IRAF講習会テキスト:

http://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo.html#iraf_prog

- Python Tutorial

<http://docs.python.jp/2/tutorial/>

- Pythonスタートブック 辻 真吾著 技術評論社