PyRAF ミニ講習会

磯貝 瑞希＠国立天文台・天文データセンター

2020 MAR 12 @国立天文台・天文データセンター

本講習会の目標

・PyRAFの使用環境を構築できるようになる

・PyRAFを対話的に使用できるようになる

・Pythonスクリプト内でPyRAFを使用できるようになる

内容

1. PyRAFのインストール (by AstroConda)
2. PyRAFの対話的使用法
3. Python言語の基礎
4. Pythonスクリプト内での使用法
5. 実習 （一次処理スクリプトの作成）（ウェブ掲載版では削除）

1. PyRAFのインストール

（by AstroConda）

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(=Space Telescope Science Institute)が開発:
  URL: http://www.stsci.edu/institute/software_hardware/pyraf
- 以下の2種類の使用モードを備える:
  - 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)
  - Pythonスクリプト内でモジュールとして利用するPythonモード
- 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
- 2019年9月でSTScIのサポートは終了したが、それ以降も(AstroConda経由での)インストール可

PyRAFのインストール (by AstroConda)

現在もPyRAFをインストール可能な「conda + AstroCondaチャンネル」を使用する

Conda:

Python用のパッケージ管理・仮想環境構築ツール(オープンソース)

本家サイト: https://docs.conda.io

クロスプラットフォームで、Windows, Mac, Linux版が提供されている。

AstroConda:

フリーなCondaチャンネルの一つ。STScIによって維持されている。

HSTなどのデータを処理・解析するために必要なソフトやツールなどを提供している。

本家サイト: https://astroconda.readthedocs.io

PyRAFのインストール (by AstroConda)

AstroCondaを利用したユーザ権限でのインストール手順は、以下の通り:

A: Condaのインストール

B: Condaの設定: AstroCondaチャンネルの登録

C: IRAF, PyRAF, ds9 のインストール

この方法では、IRAF, x11iraf, STSDAS/TABLES, PyRAF, 関連Pythonモジュールもまとめてインストールされるが、インストールされる IRAF は ver. 2.16 32bit版 のため、32bit用の各種ライブラリが必要。

以下、CentOS7でのインストール方法: （下記の「¥」は改行のエスケープ、¥なしに続けて入力しても可）

```
$ sudo yum –y install  glibc.i686  zlib.i686  ncurses-libs.i686  bzip2-libs.i686  ¥
    uuid.i686  libxcb.i686
```

→ 要ユーザパスワード入力

A: Condaのインストール 1/3

Condaの導入方法:

2つのdistribution(Miniconda/Anaconda)があり、それぞれに Python 2.7系 (=Miniconda2/Anaconda2) と Python 3.x系 (=Miniconda3/Anaconda3) が提供されている:

- Miniconda: 必要最小限のConda 管理環境を提供
- Anaconda: 完全なConda 管理環境 + 数百の有用なツール・ライブラリを(デフォルトで)提供

→ 本講習では、Python3.x系を、また待ち時間の短縮を目的として、軽量な Miniconda (=Miniconda3)を採用

Miniconda3 の入手:

Applications – Favorites – Firefox で Firefox を起動し、本家サイトのdownloadページ

URL: https://docs.conda.io/en/latest/miniconda.html

より、Linux installers の Python 3.x,「Miniconda Linux 64-bit」 を選択、表示される ダイアログで「Save File」を選択し、OKを押す

A: Condaのインストール 2/3

ダウンロードしたファイルのハッシュ値計算と確認:

$ sha256sum /home/student/Downloads/Miniconda3-latest-Linux-x86_64.sh

→ 本家サイトのハッシュ値と一致すればOK.

ダウンロードしたファイルを bash で実行: (=ユーザ権限でのインストール)

$ bash /home/student/Downloads/Miniconda3-latest-Linux-x86_64.sh

→ Enterを押すとライセンス条項が表示される。最後まで進んで、「yes」を入力

→インストール先[/home/student/miniconda3」はそのままとし、Enterを押す

→ Miniconda3 の初期化の実施: yes

→ ~/miniconda3 ディレクトリが作成される。ディレクトリのサイズは約300MB

A: Condaのインストール 3/3

確認:

$ **source ~/.bashrc**

→ condaのbase環境がactivateされる

→ コマンドプロンプトの前に、「(base)」 と表示されるようになる

デフォルトの設定では、次回以降、ログイン時に自動起動

参考: 自動起動設定の解除

$ conda config –set auto_activate_base false

$ **which conda**

→ ~/miniconda3/bin/conda

B: Condaの設定
C: IRAF/PyRAF/ds9 のインストール 1/2

B: Condaの設定: AstroCondaチャンネルの登録（以下、base環境のまま実行）

$ **conda config --add channels http://ssb.stsci.edu/astroconda**

C: IRAF/PyRAF/ds9 のインストール (本講習では 環境名: iraf37 とする）

$ **conda create -n iraf37 python=3.7 iraf-all pyraf-all ds9**

→ Proceed([y]/n)?: [Enter]

→ 問題なければ5-6分程度で終了 (環境=通信帯域幅に強く依存)

→ 終了後、~/miniconda3ディレクトリのサイズ: 約6GB（=5.4GBの増加)

C: IRAF/PyRAF/ds9 のインストール 2/2

確認

インストールした環境の有効化(アクティベート)

$ **conda activate iraf37** (conda v4.4 より conda activateを推奨)

→ コマンドプロンプト前の表示が (base) から (iraf37) に変化

$ **which cl; which pyraf; which xgterm; which ds9**

→ ~/miniconda3/envs/iraf37/bin/cl

~/miniconda3/envs/iraf37/bin/pyraf

~/miniconda3/envs/iraf37/bin/xgterm

~/miniconda3/envs/iraf37/bin/ds9

インストールした環境の無効化(ディアクティベート)

$ **conda deactivate** (conda v4.4 より conda deactivateを推奨)

補足 1/2

1: Python 2.7 環境のインストール方法：

今回 AstroCondaでインストールした PyRAF は Python 3.x 系をベースにしているが、

AstroCondaの本家サイトのFAQ

https://astroconda.readthedocs.io/en/latest/faq.html

では、Python 2.7系の使用を推奨している

（「STSDASのPythonコードが Python 2.7 とそれ以前のバージョンを特に

ターゲットにしているため」とのこと）

Python 2.7 環境をインストールする場合は、本テキストと以下の2点が異なる:

- A: Miniconda2/Anaconda2 をインストール
- C: conda create -n iraf27 python=2.7 iraf-all pyraf-all ds9

補足 2/2

2: IRAF 32bit版がインストールされる理由:

本家サイトのFAQによれば、「多くのタスクで64bit版バイナリを用意するには、

ソースコードの大幅な変更が必要だったため」とのこと

2. PyRAFの対話的使用法

PyRAFの対話的使用とは?

- PyRAFが備える2種類の使用モードのうちの一つで、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)の使用のこと

- 対話モードは、IRAFのCLとほぼ同じように利用可能

  → 最初にIRAF CLとの違いを、次に使用法を紹介

IRAF CLとの違い 1/3

- PyRAFは独自のグラフ描写カーネルを持つ
  → xgterm不要 （AstroCondaでのインストールでは、xgtermもインストールされるが…）

- 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 -> clPrint, 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: PYimport

IRAF CLとの違い 2/3

- 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ではパッケージのアンロードが出来ない

  bye, keepコマンドは存在するが、実際には何もしない

- 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

IRAF CLとの違い 3/3

- 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()

← CLスクリプトをPyRAFで実行する場合の話

PyRAF使用準備

PyRAFを起動する前に、IRAFのlogin.clを作成しておく:

$ mkdir ~/iraf

$ cd ~/iraf

$ mkiraf

→ Enter terminal type [default xterm-256color]: xterm

（デフォルトのターミナル設定ではPyRAF起動時にエラーが出て起動不可。

xtermがインストールされていなくても問題なし。xtermの代わりにxgtermでも良い）

$ cd

→ ~/irafディレクトリ以下にlogin.clを作成しておくと、どのディレクトリからでも

PyRAF起動時に~/iraf/login.clを参照する

PyRAF対話モードの使い方 1/2

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、iraf37環境をactivateしてから

$ 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 EXPORT Thu May 24 15:41:17 MST 2012
    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.

 The following commands or packages are currently defined:

 (Updated on 2013-12-13)

clpackage/:
 adccdrom/     esowfi/       mem0/         rvsao/        user/
 cfh12k/       finder/       mscdb/        softools/     utilities/
 cirred/       fitsutil/     mscred/       sqiid/        vo/
 clpackage/    gemini/       mtools/       stecf/        xdimsum/
 ctio/         gmisc/        nfextern/     stsdas/       xray/
 cutoutpkg/    guiapps/      noao/         system/
 dataio/       images/       obsolete/     tables/
 dbms/         language/     plot/         ucsclris/
 deitab/       lists/        proto/        upsqiid/
PyRAF 2.1.15 Copyright (c) 2002 AURA
Python 3.7.6 Copyright (c) 2001-2019 Python Software Foundation.
Python/CL command line wrapper
  .help describes executive commands
-->
```

← PyRAF対話モードのプロンプト

PyRAF対話モードの使い方 2/2

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

PyRAFの起動に IRAFのlogin.clは必須ではないが、用意しておくとPyRAF起動時にIRAF起動時と同じ環境(=login.cl内の変数設定、登録タスク・シェルコマンド、パッケージロード)で使用可

試してみよう 1/2

以下を実行してみよう:

--> !ds9&

--> displ dev$pix 1

--> 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 2/2

--> 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」でタスク名の補完は不可だが、変数名の補完が可能

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

→ imu.imstat[tab]とすれば、タスク名の補完も可能

→ iraf.imu.imstatistics と補完される

EPAR 変数エディタ 1/2

- **epar タスク名で変数編集GUIが起動する**

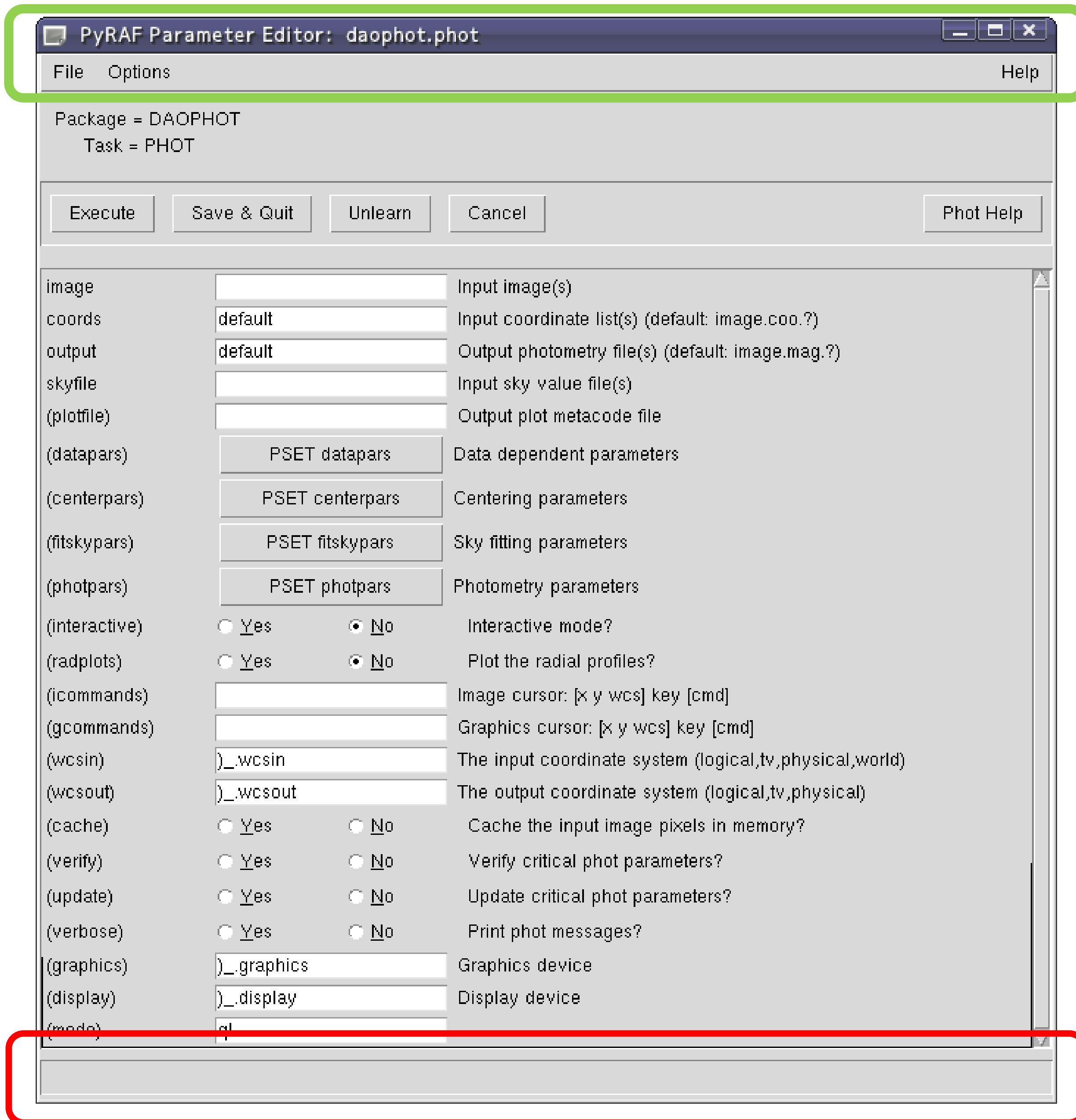
  例: daophot.phot

  --> **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:

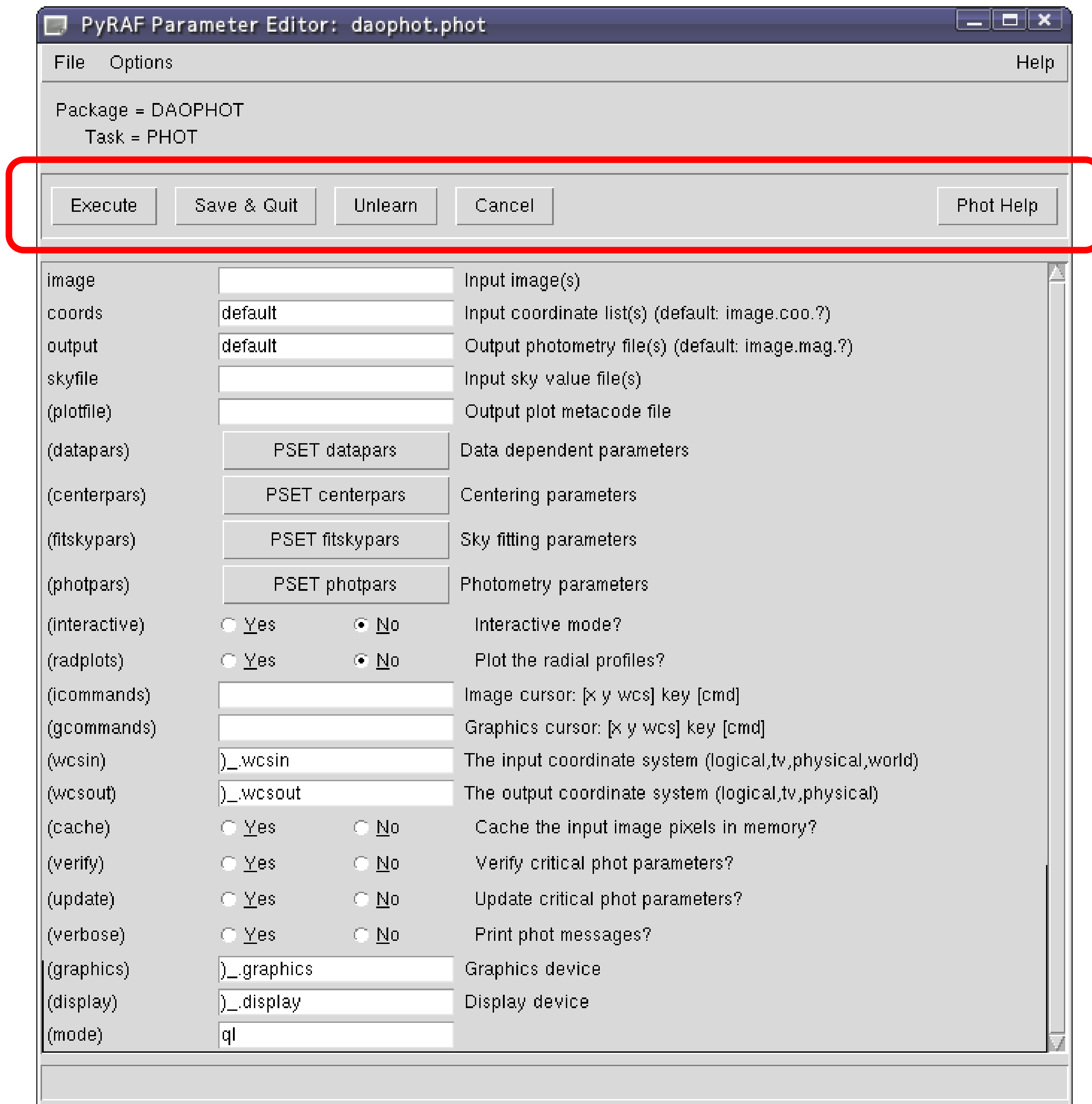
File: アクションボタン機能を直接選択可能

Options: ヘルプの表示方法を選択可能:
(ウィンドウ 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を別ウィンドウで表示

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

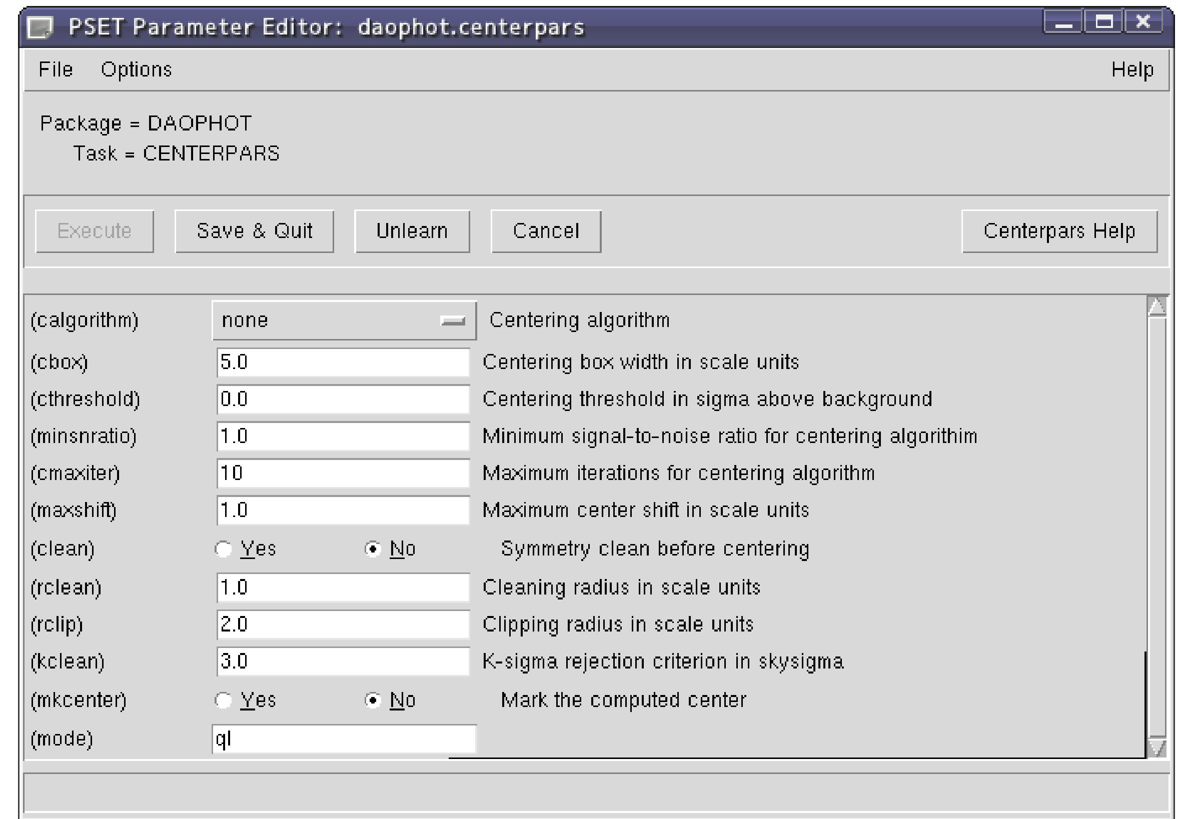
PyRAF Parameter Editor: daophot.phot

File Options Help

Package = DAOPHOT
Task = PHOT

Execute | Save & Quit | Unlearn | Cancel | Phot Help

Parameter	Value	Description
image		Input image(s)
coords	default	Input coordinate list(s) (default: image.coo.?)
output	default	Output photometry file(s) (default: image.mag.?)
skyfile		Input sky value file(s)
(plotfile)		Output plot metacode file
(datapars)	PSET datapars	Data dependent parameters
(centerpars)	PSET centerpars	Centering parameters
(fitskypars)	PSET fitskypars	Sky fitting parameters
(photpars)	PSET photpars	Photometry parameters
(interactive)	○ Yes ◉ No	Interactive mode?
(radplots)	○ Yes ◉ No	Plot the radial profiles?
(icommands)		Image cursor: [x y wcs] key [cmd]
(gcommands)		Graphics cursor: [x y wcs] key [cmd]
(wcsin)	)_.wcsin	The input coordinate system (logical,tv,physical,world)
(wcsout)	)_.wcsout	The output coordinate system (logical,tv,physical)
(cache)	○ Yes ○ No	Cache the input image pixels in memory?
(verify)	○ Yes ○ No	Verify critical phot parameters?
(update)	○ Yes ○ No	Update critical phot parameters?
(verbose)	○ Yes ○ No	Print phot messages?
(graphics)	)_.graphics	Graphics device
(display)	)_.display	Display device
(mode)	ql	



EPAR変数エディタ 2/2

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)

(ディレクトリブラウザ)

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利用する

（一部の機能はIRAFのグラフィックカーネルを使用する．例: 印刷）

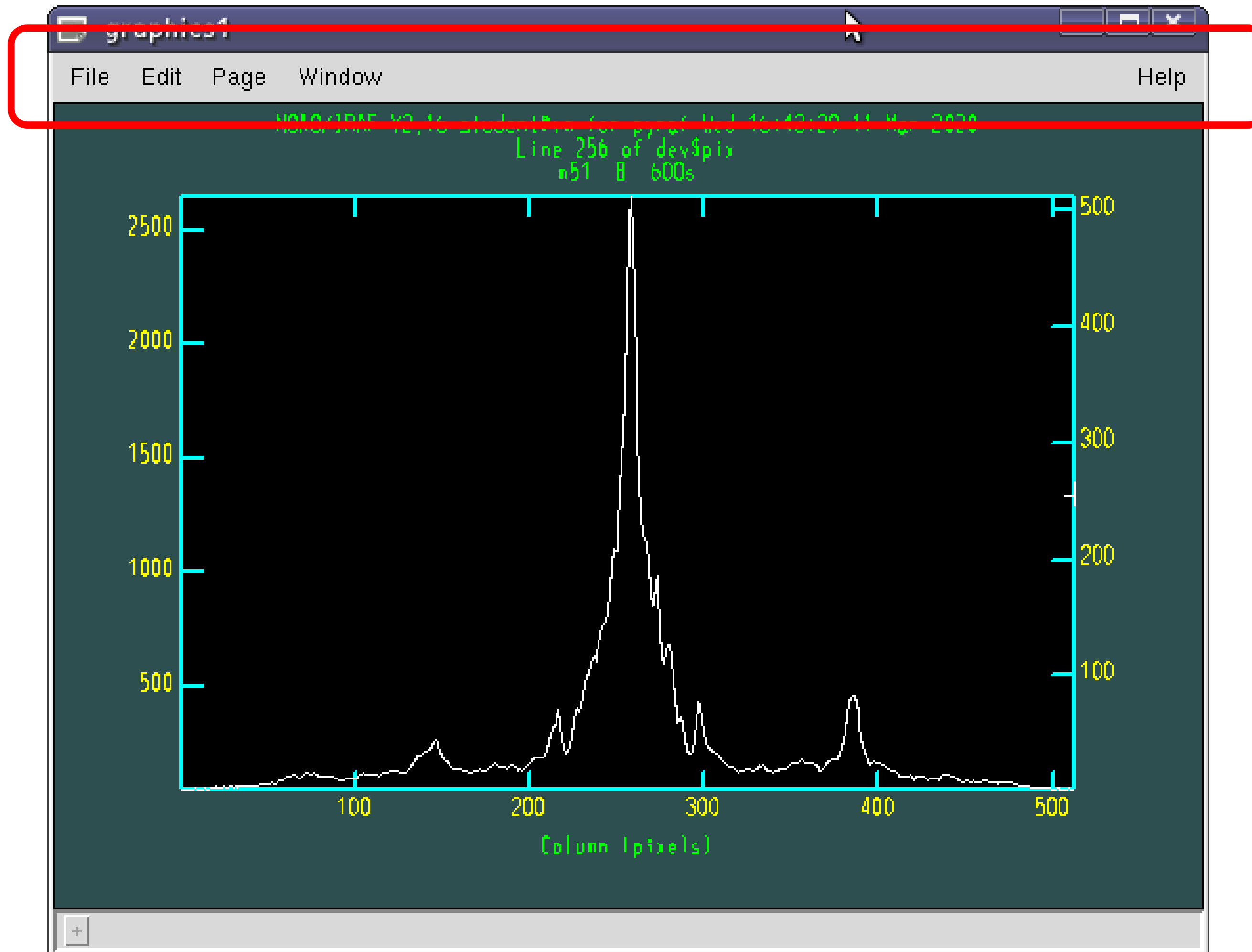
・IRAFと完全に同一ではないが、多くの同等機能を持つ

・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

・対話機能も利用可能。ただし、大文字のキー入力は一部未対応

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他

Edit: Undo, Redo, Reflesh, Delete他

Page: Next, Back, First, Last

Window: New他

（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)

- Python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

・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

PyRAFでの画像描写

- tv.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カーネルを通して実行される

Pythonモードでの使用法 1/2

・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 2/2

・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')

実行例:

```
stsdas
iraf.imcalc(PYin='dev$pix', out='out.fits', eq='log10(im1)', pixt='double')
```

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!

3. Python言語の基礎

Python言語の特徴

- 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（ブロック最後の空行追加を推奨）
- 豊富な拡張機能が標準ライブラリ・外部ライブラリとしてモジュール形式で提供されている
- 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
- 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: 3.x系を使用（ver. 3.7.6）

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
7. 辞書(dictionary） キー:値 ペアのリスト

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をリスト型の変数名＋大括弧[]で囲むことで指定可能

例: data[0]　→ ’a’

整数・小数型

四則演算:

和: a + b

減: a - b

積: a * b

商: a / b

べき乗: x**y, pow(x, y)

文字列操作

- 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']

リスト操作

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

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

組み込み関数: print(), len()

print(): 出力 (← Python 3.xでprint文からprint関数に)

例: print(string) → 改行付きでstringの内容を表示

(改行なし表示: print(string, end=''))

形式を指定した出力:

```
print('a: {0:10s} b:{1:7.5f}'.format(a,b))
```

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

リストの先頭要素sys.argv[0]はスクリプトファイル名

使用法:

```
import sys
print(sys.argv[1:])
```

シェルコマンドの実行

subprocessモジュールのrunメソッドを使用する

（Python 2.xを含む3.4以前では、runメソッドがないのでcallメソッドを使用する）

使用法:

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

例:

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

ファイル読み書き 1/2

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+')

ファイルからの読み込み(推奨方法):

```
for line in f:
    print(line)
```

ファイルから1行ずつ読み出し、ループ内で処理をする

ファイル読み書き 2/2

ファイルへの書き込み:

f.write(string)

print(string, file=f)

stringの内容をファイルに書き込む

write()で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

例:

```
i=2
f.write(i)
```

→ TypeError: write() argument must be str, not _io.TextIOWrapper

```
f.write(str(i))
```

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	a == b	
aとbは不一致	a != b	
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	
aはbに含まれる	a in b	（bは文字列やリストなど）
aはbに含まれない	a not in b	（bは文字列やリストなど）

関数の作成

関数の宣言: def

```
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)
```

引数にデフォルト値を設定する場合:　引数=値　とする

関数の使用

関数を使う方法:

関数名(引数1, ...)

例:

myfunc()	引数なしで実行
myfunc(list)	1つの引数で実行
myfunc(list,a,b)	3つの引数で実行
y = myfunc(list,a,b)	関数の結果を変数yに代入

クラスの作成(例)

複数のデータを1個の変数で保持できると便利である

ここではそのような新しいデータ型(=クラス)の作成例を紹介する

以下は、複数のデータ型を持つクラスを定義する例である（selfは必須）

属性の定義のみで、メソッドは定義していない

```
class   クラス名:                                  # クラス名は()をつけない
        def   __init__(self, 引数1, 引数2):      # インデント
                self.属性1  = 引数1
                slef.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:
    def  __ini t __(self, arg1, arg2):
            self.x  = arg1
            self.y  = arg2

a = scat(data1, data2)
print(a.x, a.y)
```

4. Pythonスクリプト内での使用法

Pythonスクリプトでの記述

PyRAF:

2種類の使用法:

- 対話モード
- Pythonモード

Pythonスクリプトでは「Pythonモード」で記述する

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、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 1/2

PythonスクリプトでのPyRAFの使用手順:

1. pyraf.irafモジュールをimportする

   例: from pyraf import iraf

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

   例: iraf.daophot() (← 引数なしの場合でも「()」が必要)

   ← from iraf import dapho

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

PythonスクリプトでのPyRAFの使用 2/2

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 1/2

Pythonでは IRAFで利用可能な pipe「|」やリダイレクト(「<」,「>」)を使用できない

→ 代わりに特殊なタスク変数「Stdin」,「Stdout」,「Stderr」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 2/2

補足:

- Stdin, Stdout, Stderr 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

- Stdout を指定せず、Stderrのみ指定した場合には、標準出力の内容もStderrに出力される

  （両方を指定した場合には、Stderrには標準エラー出力の内容のみ出力される）

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

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.lParam()

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(...)
```

複数のグラフウィンドウの利用

Pythonスクリプト内でグラフウィンドウを操作可能

ウィンドウのアクティブ化:

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

Pythonスクリプトの基本構文 1/2

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 #7 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 2/2

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
- #8: *sys.argv[1:3]　*「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

Pythonスクリプトの実行

実行方法は2通り:

1. シェルからコマンドとして実行:

   $ /path_to_scripts/スクリプト名　引数1 引数2 ...

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 → import myscript, myscript.myfunc()

   (用法注意: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)

PyRAF+Python参考文献

PyRAF+Python参考文献

本資料の作成にあたり、以下を参考にしている

・ The 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

・ PyRAF FAQ

http://www.stsci.edu/institute/software_hardware/pyraf/pyraf_faq

・ Python Tutorial

https://docs.python.org/ja/3.7/tutorial/