講習会後改訂+ウェブ掲載版　2020.10.30

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 OCT 28-29 @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スクリプトでのPyRAFの使用法
5. Pythonスクリプト実習　（一次処理スクリプトの作成）←ウェブ掲載版では割愛

本講習ではPython言語の説明を必要最小限(+α)に限定
実習にはデータ整約において共通の処理である一次処理を採用

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 (リンク切れ)
- 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とIRAFの両方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。

参考: CentOS7/8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/dnf –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

今回は多波長解析システムを使用・導入済みのため不要

情報元: https://astroconda.readthedocs.io/en/latest/faq.html#why-is-iraf-32-bit-instead-of-64-bit

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:

左上のActivitiesを選択 → 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 3.8, 「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

注意!
2020/10/23現在、Python 3.x系の最新版は3.8。一つ下の3.7は32bit版(利用不可)

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 ~/Downloads/Miniconda3-latest-Linux-x86_64.sh

→ 本家サイトのハッシュ値(879457...daf688)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 ~/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[/homeXX/account/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350MB

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 と表示されればOK

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）

（補足: python=3.8を指定した場合、pyraf-allが3.8に対応しておらず、エラーとなりインストールに失敗する）

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 python; which mkiraf; which ds9　(一つずつ実行でも可)

→ ~/miniconda3/envs/iraf37/bin/cl

~/miniconda3/envs/iraf37/bin/pyraf

~/miniconda3/envs/iraf37/bin/python

~/miniconda3/envs/iraf37/bin/mkiraf

~/miniconda3/envs/iraf37/bin/ds9

システムインストール版のパス:
/usr/local/iraf2161/bin/cl
/usr/local/python/2.7/bin/pyraf
/usr/local/python/2.7/bin/python
/usr/local/iraf2161/bin/mkiraf
/usr/local/bin/ds9
が表示された場合には、次ページを参照

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)

トラブルシュート

前ページでシステムインストール版のパスが表示された場合:

conda deactivateを2回実行し、プロンプトの前に(base)が表示されなくなったことを確認の後、

conda activate iraf37を実行

```
(iraf37) $ conda deactivate
(base) $ conda deactivate
$ conda activate iraf37
(iraf37) $ which cl;  which pyraf; which python; which mkiraf; which ds9
```

以上でも解消しなければ、以下をconda実行用ターミナルで実行する（システムインストール版のパスを無効化）

```
$ PATH=$(echo $PATH | sed -e "s/:/¥n/g" | egrep -v 'iraf|python|/usr/local/bin' | xargs | sed "s/ /:/g")
```

（¥はバックスラッシュ）

注意! ターミナル毎に実行する必要あり

本講習会用の設定

ターミナルを起動した時点でiraf37環境が有効となるよう、環境設定ファイル ~/.bashrc に iraf37環境のactivate用の設定を記述しておく （弊害もあるので、常時この設定を使用することは非推奨）

テキストエディタで ~/.bashrc を開き、ファイルの末尾に以下を記述:

```
conda deactivate
conda deactivate
conda activate iraf37
which cl;  which pyraf; which python; which mkiraf; which ds9
```

補足:
conda deactivateを2回実行しているのは、
確実にbase環境をdeactivateするため

環境設定ファイルを読み込み、p.11と同じコマンドパスが表示されることを確認する

```
$ . ~/.bashrc
  → ~/miniconda3/envs/iraf37/bin/{cl, pyraf, python, mkiraf, ds9}
    (iraf37) $
```

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

情報元: https://astroconda.readthedocs.io/en/latest/faq.html#why-is-iraf-32-bit-instead-of-64-bit

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

← 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:

ターミナルを立ち上げ、iraf37環境のactivateを確認してから

$ 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環境のactivateを確認してから

$ 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 xterm...

  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 3.7.9 Copyright (c) 2001-2020 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-**　　補足: for- : format-の略

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

--> **imstat dev$pix for- fi=mid**　　補足: mid : midptの略

→ 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）

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**

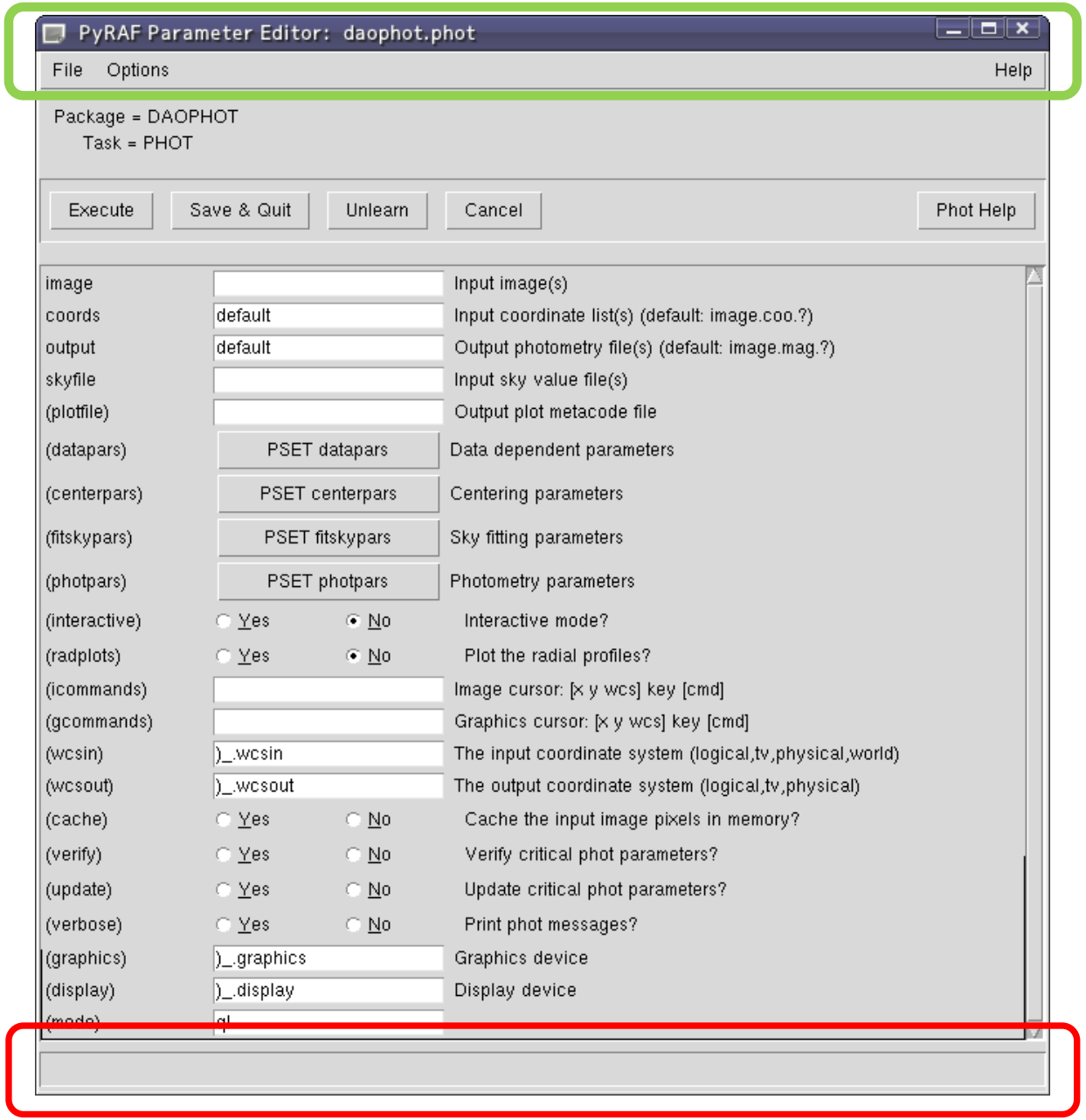
  例: daophot.phot

  --> **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:

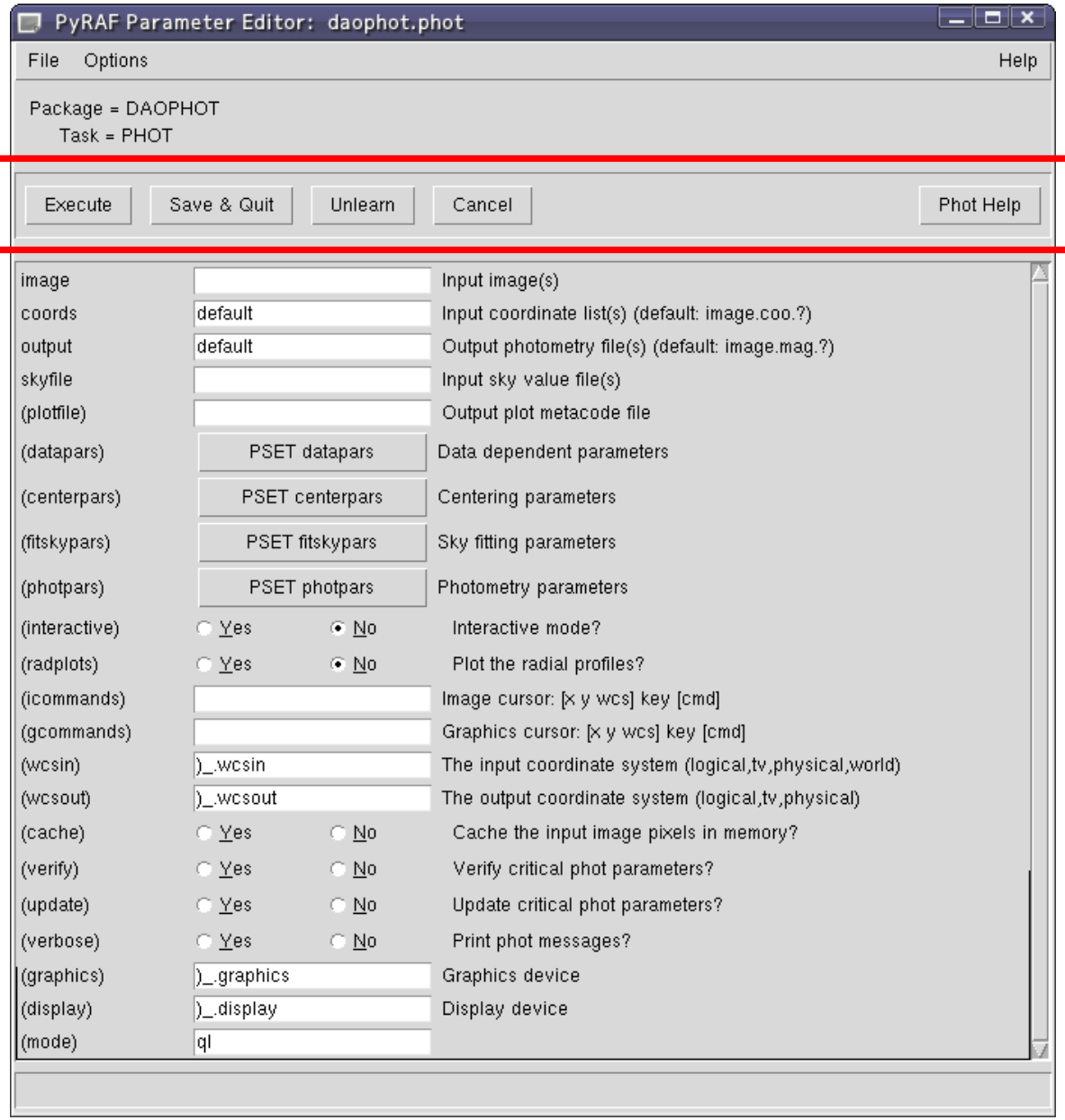
File: アクションボタン機能を直接選択可能

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

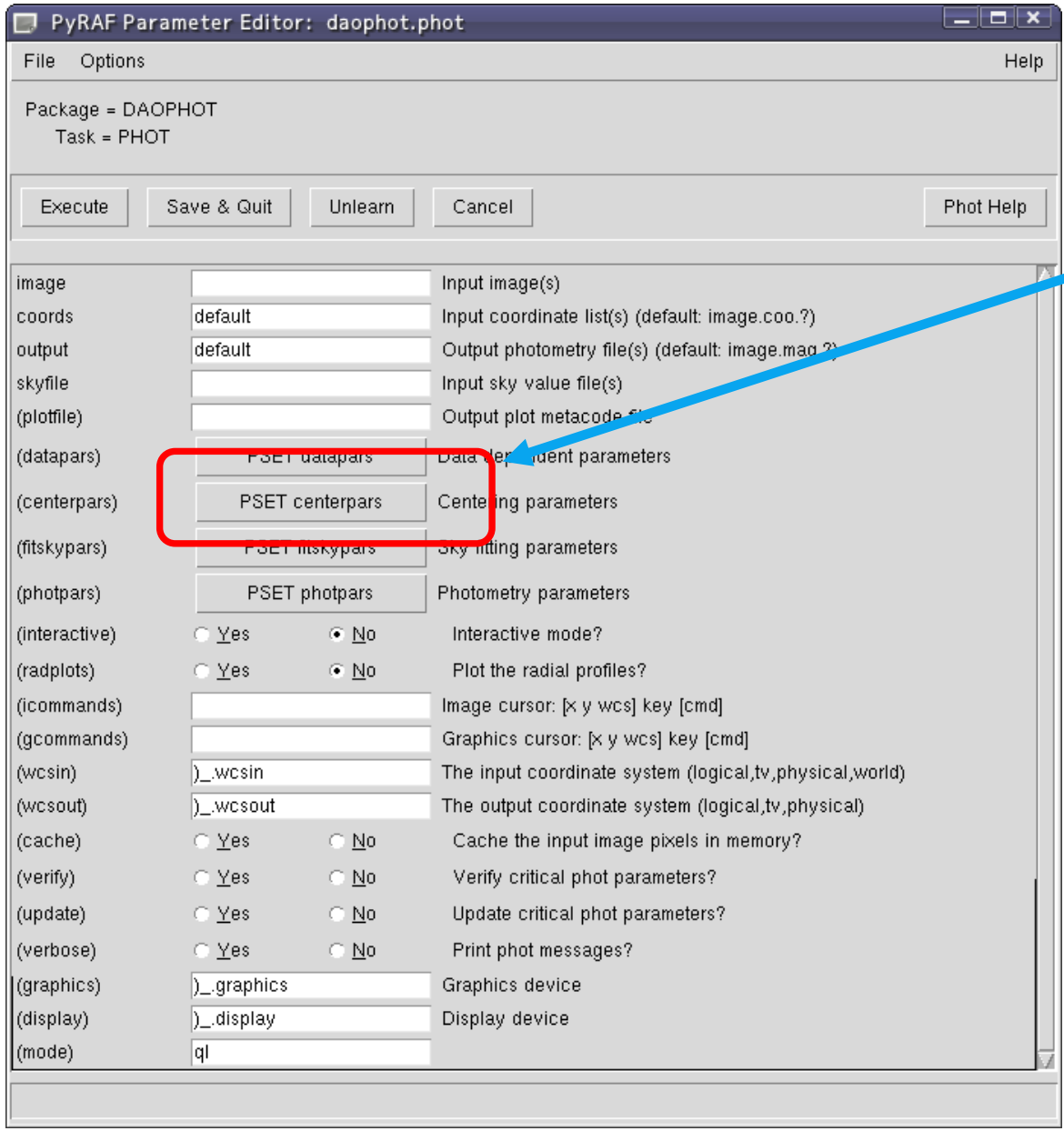
Save&Quit: 現在の変数値を保存して終了

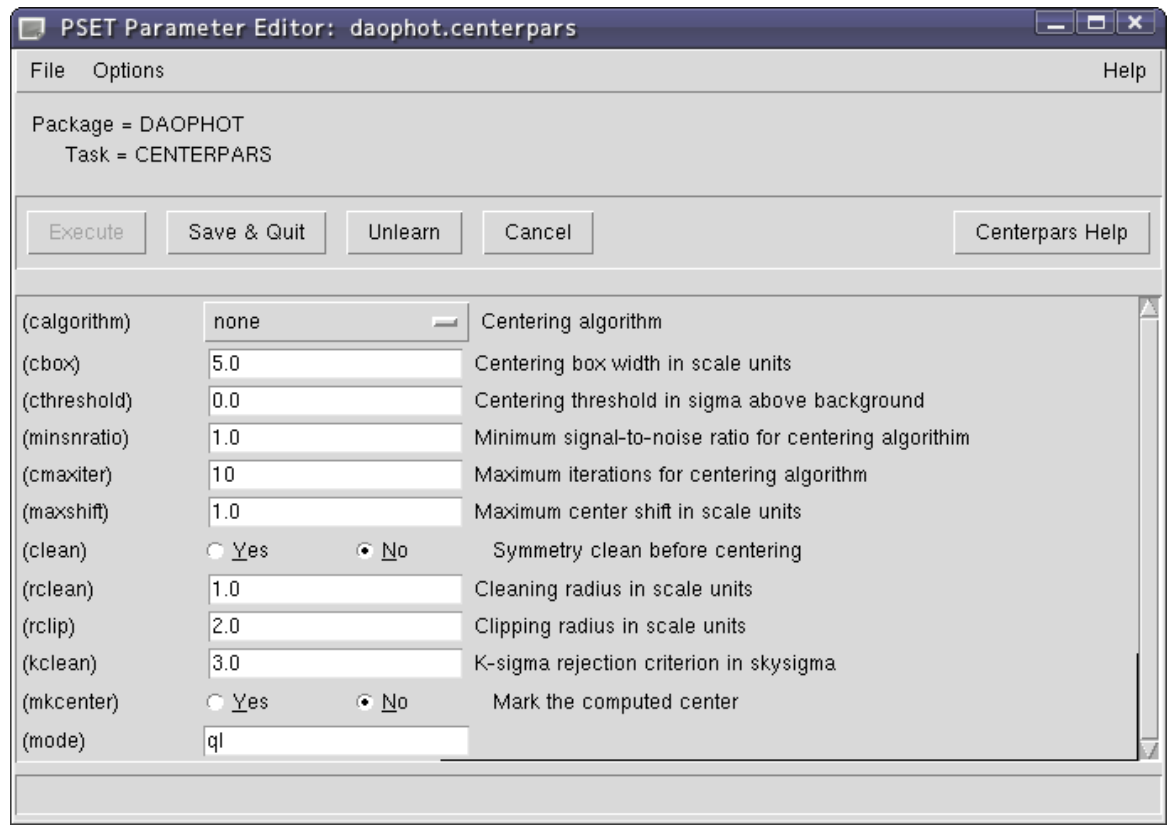
Unlearn: すべての変数値をシステムのデフォルト値に戻す

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





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）

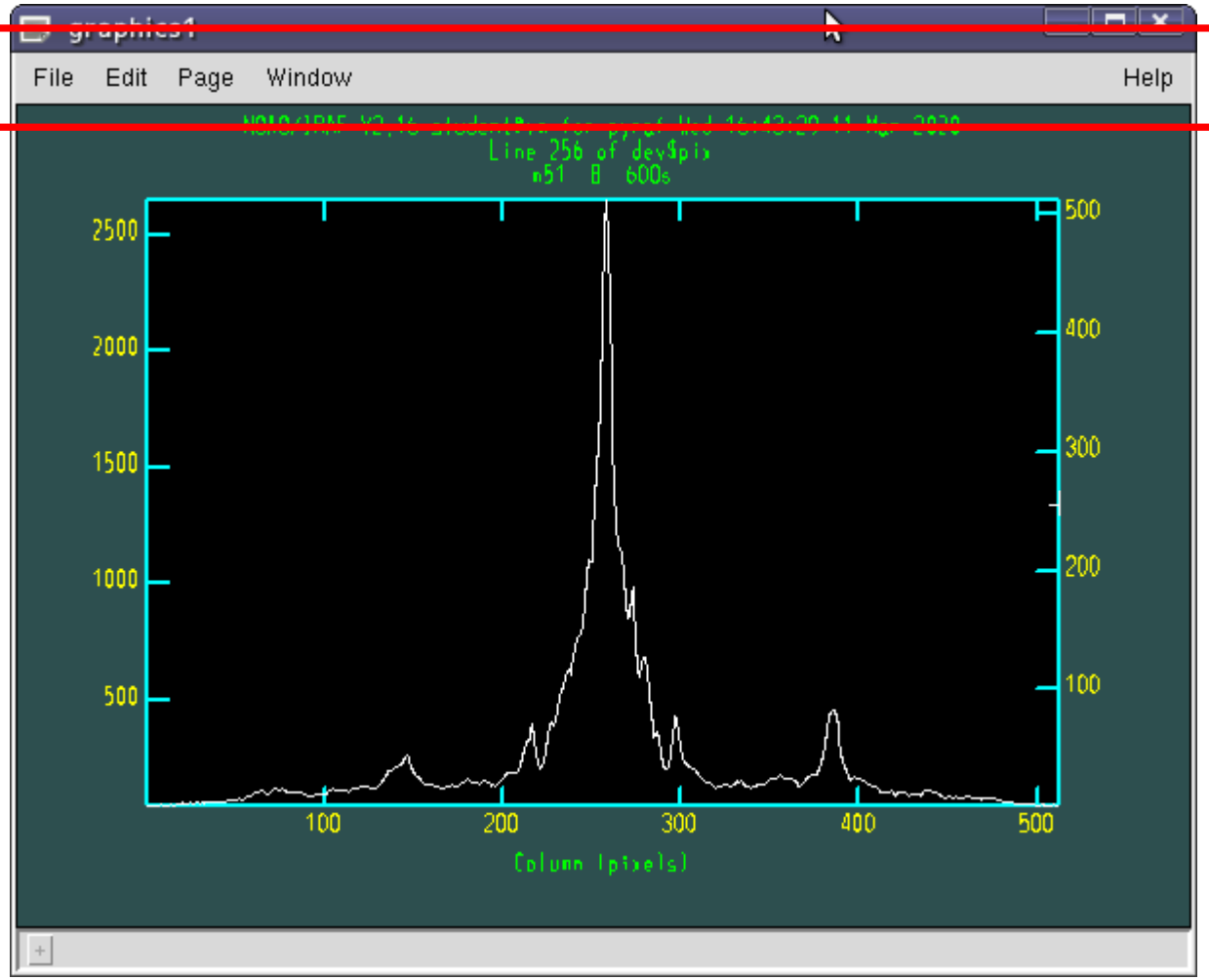
・**IRAFと完全に同一ではないが、多くの同等機能を持つ**

・**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

補足:

・implotの終了:「q」キーを押す(赤線の十字が消え、ターミナルが入力受付状態に戻る

・グラフウインドウの消去: implotを終了した後に、File – 「Quit Window」 または「x」ボタン

注意: File - 「Close Window」で消去した場合、次にグラフウインドウを表示した際にウインドウが表示されない
(Activitiesで一覧を表示すればグラフウインドウも表示される)

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’)

(補足: yes, noは’ ’で囲まなくても使用可)

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!

参考: 対話モードの場合(p.24-25)

```
displ   dev$pix  1
imhe    dev$pix
imhe    dev$pix  l+
imstat  dev$pix
imstat  dev$pix  for-
imstat  dev$pix  for-  fi=mid
```

3. Python言語の基礎

Python言語の特徴　1/2

- **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　(詳細は次ページ以降)
- **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. 3.7.9）

例: 2.x系と3.x系の違い(の一例):

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

Python言語の特徴　2/2

・**対話モードあり**

→ スクリプトの記述や実行結果などを簡単に確認することが可能

（テキストに出てくる記述例を実際に実行して結果を確認するなど）

起動方法:

```
$ python
```

→ Python 3.7.9 (default, Aug 31 2020, 12:42:55)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>　←入力受付プロンプト

終了方法:

```
>>> exit()
```

拡張機能の利用: import 1/2

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 2/2

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**

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

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[-1]　→ ’c’

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)

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　1/2

- **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　2/2

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

類似メソッド:
str.lstrip('chars') 先頭文字列を削除
str.rstrip('chars') 末尾　〃　を削除

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: 'object.d.fits'.replace('d.fits','m.fits')　→ 'object.m.fits'

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

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

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

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))

aは文字列を格納した変数
bは実数を格納した変数

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

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

→　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(スクリプト名)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)
```

ファイルの存在確認と削除

ファイルの存在確認: os.pathモジュールのos.path.isfile()メソッドを使用

ファイルの削除: osモジュールのos.remove()メソッドを使用

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

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)
```

ファイル読み書き　1/3

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')　　# fは任意の変数

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 2/3

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))
```

ファイル読み書き 3/3

ファイルのクローズ:

```
f.close()        # ファイルオブジェクト.close()
```

書き込みの場合、ファイルをクローズして初めてファイルが生成される。
スクリプト中で書き込みしたファイルを使用したい場合には、使用前に
ファイルクローズを忘れずに実行しておく必要がある。

例: ファイルへの書き込み一連の処理:

```
f=open('myfile.txt', 'w')
print('str: {0} value:{1}'.format(str,value),file=f)
f.close()
```

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）

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、こちらは対話用のため
スクリプトでの使用を避けるべき）

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):              # 引数の数が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

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。

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(__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.属性1  = 引数1              # 引数1を属性1に格納
                    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:
    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PyRAF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スクリプト内での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)

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

   (Pythonスクリプト内では yes, no も' 'で囲む必要あり)

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（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

→ 確認をしてくるので[enter]を2回押す

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: lParam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(UTF-8使用):

```
#! /usr/bin/env python                    # 1

import sys                                # 2
from pyraf import iraf                    # 3

def mytask(arg1,arg2):                    # 4
    iraf.imstat(arg1, format='no',fields=arg2)  # 5
    ...

if __name__ == "__main__":                # 6
    mytask(*sys.argv[1:3])                # 7
```

#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: sysモジュールのimport

#3: pyraf.irafモジュールのimport

#4:「自作タスク」関数の定義

#5: irafタスクの記述

#6: 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

#7: 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
- #4: 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
- #6: シェルから実行可能とする場合のみ必要
- #7: *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:

本講習では1を採用

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　1/2

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

リンク切れ(http://www.stsci.edu/institute/software_hardware/pyraf/pyraf_faq)

・ Python Tutorial

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

PyRAF+Python参考文献　2/2

・ 天文データセンター講習会: 過去の講習会資料 (IRAF/PyRAF講習会)
https://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo.html#iraf_prog

なお、本資料は2015年度実施の「IRAF/PyRAF講習会」や2017年度と2018年度実施の「IRAF/PyRAFインストール講習会」の講習資料をベースとしています。
興味がありましたら、ぜひ過去の講習会資料と合わせてご参照ください。