Advanced Astronomical Observations 2021
Session 01: Basic Python Programming

Kinoshita Daisuke

24 February 2021
publicly accessible version

About this file...

- Important information about this file
  - The author of this file is Kinoshita Daisuke.
  - The original version of this file was used for the course "Advanced Astronomical Observations" (course ID: AS6005) offered at Institute of Astronomy, National Central University from February 2021 to June 2021.
  - The file is provided in the hope that it will be useful, but there is no guarantee for the correctness. Use this file at your own risk.
  - If you are willing to use this file for your study, please feel free to use. I'll be very happy to receive feedback from you.
  - If you are willing to use this file for your teaching, please contact to Kinoshita Daisuke. When you use this file partly or entirely, please mention clearly that the author of the original version is Kinoshita Daisuke. Please help me to improve the contents of this file by sending your feedback.
  - Contact address: `https://www.instagram.com/daisuke23888/`

This course, "Advanced Astronomical Observations", aims to provide an opportunity to learn Python programming for data reduction and analysis of observational data. Topics for this course include, but not limited to,

- manipulation of FITS files,
  - reading and modifying keywords in header,
  - reading and writing image data,
- characterisation of properties of CCD imager using bias, dark, and flatfield images,
  - stability of bias,
  - dark current generation rate,
  - gain (A/D conversion factor) and readout noise,
- standard CCD data reduction,
  - dark subtraction,
  - flatfielding,
- improvement of WCS-related keywords in FITS header,
- aperture photometry
  - construction of PSF profile,
  - photometry of photometric standard stars,
  - atmospheric extinction coefficients,
  - solving transformation equation,

- estimate of sky background brightness,
- estimate of limiting magnitude (detection limit),
- differential photometry and construction of photometric lightcurve,
  - construction of intra-night lightcurve,
  - construction of inter-night lightcurve,
  - period search and construction of folded lightcurve,
- three-colour composite of multi-band data,

1 Suggested operating systems

Many astronomical software is developed on Unix-like operating systems. For this course, use of BSD or Linux operating system is highly encouraged. Here is a list of suggested operating systems.

- NetBSD: `https://www.netbsd.org/`
- FreeBSD: `https://www.freebsd.org/`
- Debian GNU/Linux: `https://www.debian.org/`

If you are not using a Unix-like operating system, and you are not willing to destroy your favourite operating system, you may consider to use a hypervisor to install either NetBSD, FreeBSD, or Debian GNU/Linux on a virtual machine.

2 Programming language

The official programming language for this course is Python. To learn about Python programming language, please visit following website. (Fig. 1)

- Python: `https://www.python.org/`

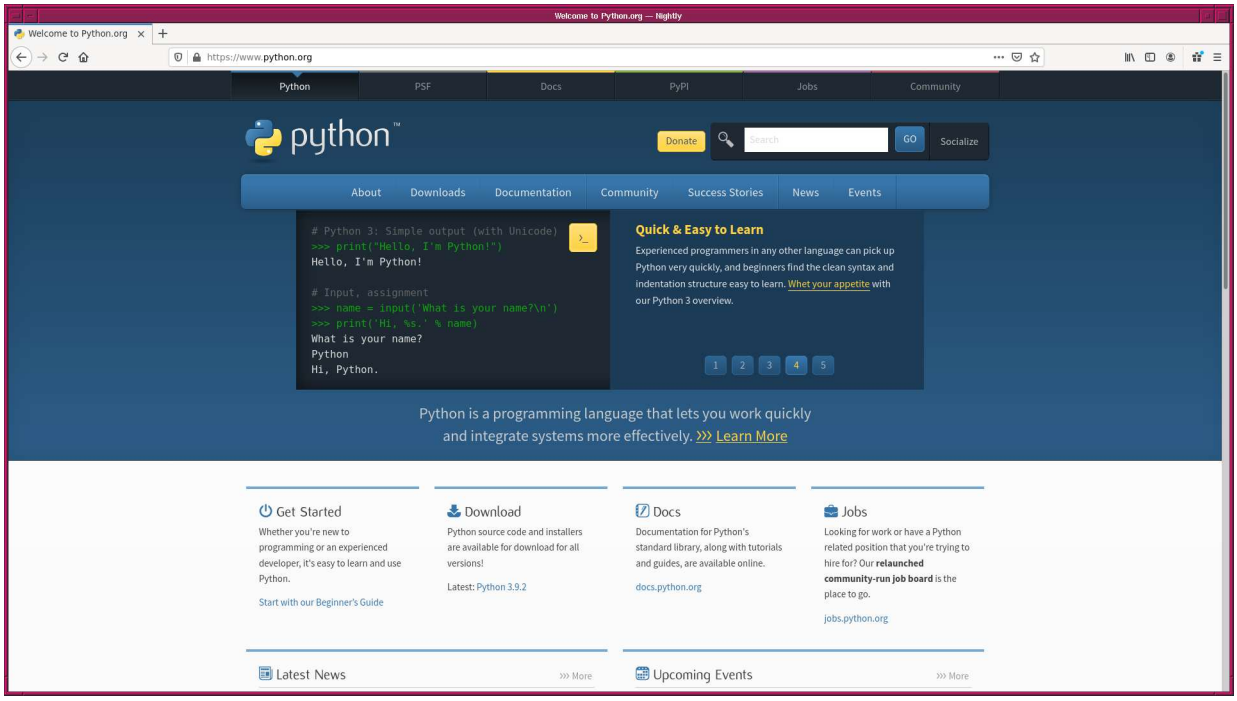

Figure 1: The official website of Python.

We use Python 3 for our programming for this course. Install Python 3 on your computer. For the installation of Python 3, you may consider to use

- pkgsrc system on NetBSD,
- ports collection on FreeBSD,
- APT package management system on Debian GNU/Linux.

The official documents for Python 3 can be found at following website.

- `https://docs.python.org/3/`

The latest version of Python is 3.9.2 at the time of this writing (23 February 2021). Make sure to install Python 3.9.2 on your computer.

2.1 Text editor

For this course, you are encouraged to write Python scripts using a text editor. The text editor "GNU Emacs" is highly recommended.

- GNU Emacs: `https://www.gnu.org/software/emacs/`

The GNU Emacs has strong supporting functions for writing computer programs.

2.2 Terminal emulator

When you finish writing a Python script, you run the script on a terminal emulator. The use of "xterm" is suggested.

3 Suggested readings

Here is a list of suggested readings. If you are new to Python programming, you are encouraged to read these books.

- "Python Tutorial", `https://docs.python.org/3/tutorial/`.
- "Learning Python" 3rd Edition, 2008, Mark Lutz, O'Reilly, `http://shop.oreilly.com/product/9780596513986.do`.
- "Programming Python" 4th Edition, 2010, Mark Lutz, O'Reilly, `http://shop.oreilly.com/product/9780596158118.do`.
- "Python Cookbook" 2nd Edition, 2008, David Ascher, Alex Martelli, Anna Ravenscroft, O'Reilly, `http://shop.oreilly.com/product/9780596007973.do`.

"Python Tutorial" is highly recommended, if you have just started to write Python scripts. It is available at Python official website.
"Library Reference" of Python can be found at following web page, and it is very useful.

- `https://docs.python.org/3/library/index.html`

4 Python modules used for this course

Following Python modules are used for this course. Visit the official websites of these modules, and install them on your computer.

- NumPy
  - `https://numpy.org/`
- SciPy
  - `https://scipy.org/`
- Matplotlib
  - `https://matplotlib.org/`

- Astropy
  - `https://www.astropy.org/`
- Astroquery
  - `https://astroquery.readthedocs.io/`
- ccdproc
  - `https://ccdproc.readthedocs.io/`
- photutils
  - `https://photutils.readthedocs.io/`
- gwcs
  - `https://gwcs.readthedocs.io/`
- ginga
  - `https://ginga.readthedocs.io/`

The documentation of above modules are available on the official website. Read official documents and learn about them.

5 Making and executing a Python script

Here is a simple example of making and executing a Python script.

5.1 Finding your Python executable

If you are using Python 3.9, then try following command to find the location of Python executable.

```
% which python3.9
/usr/pkg/bin/python3.9
```

We now know that Python executable is located at "`/usr/pkg/bin/python3.9`".
If you are using Python 3.8, then try following.

```
% which python3.8
/usr/pkg/bin/python3.8
```

If you do not have an executable of given name, then you see following message.

```
% which python3.7
python3.7: Command not found.
```

The name and location of Python executable depends on the operating system. It may be `/usr/bin/python` or `/usr/local/bin/python3.9`. Make sure to know where do you find Python executable on your computer.

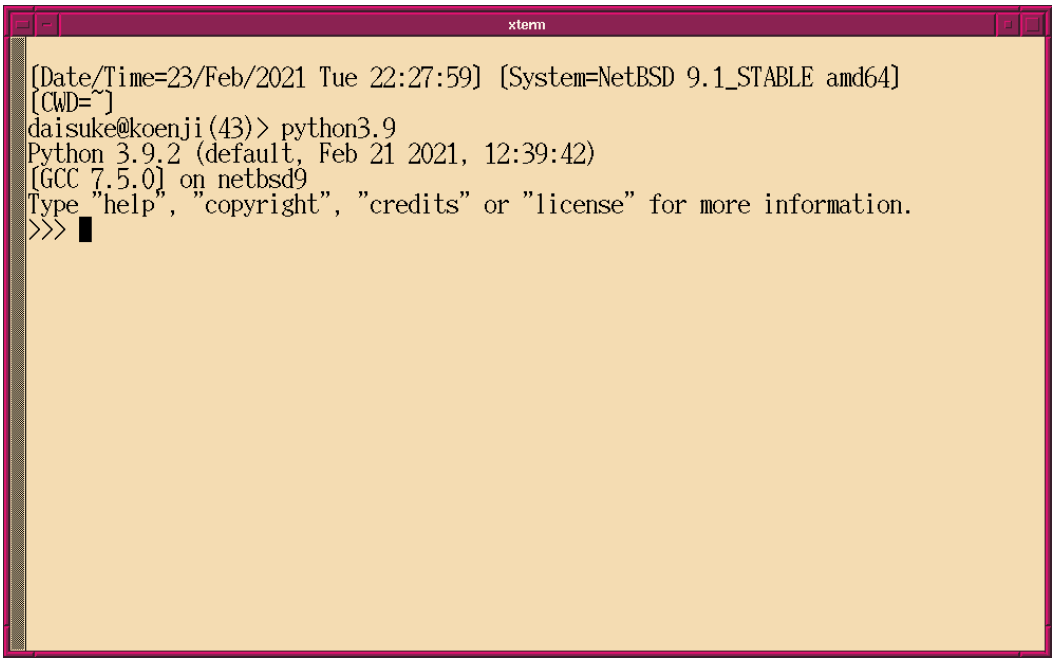

Figure 2: Python in interactive mode.

5.2 Starting Python in interactive mode

Once you find the location of Python executable, start Python in interactive mode. Try following if your Python executable is `python3.9`. (Fig. 2)

```
% python3.9
Python 3.9.2 (default, Feb 21 2021, 12:39:42)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To leave from Python interactive mode, type `exit ()` or Ctrl-D.

```
% python3.9
Python 3.9.2 (default, Feb 21 2021, 12:39:42)
[GCC 7.5.0] on netbsd9
Type "help", "copyright", "credits" or "license" for more information.
>>> exit ()
```

5.3 Making a simple Python script

Use your favourite text editor to prepare a simple Python script of following content. (Fig. 3)

Python Code 1: ao2021_s01_01.py

```
#!/usr/pkg/bin/python3.9

print ("This is a very simple Python 3 script.")
```

Save this script as “`ao2021_s01_01.py`”.

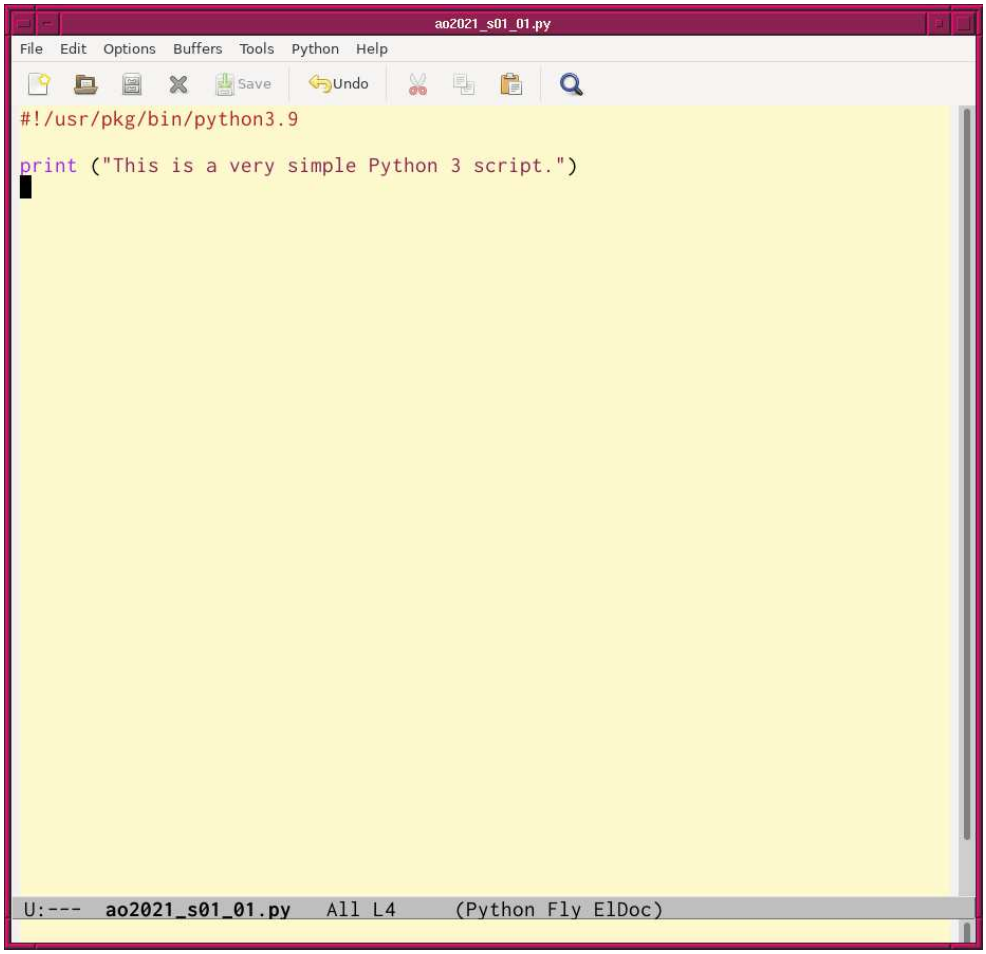

Figure 3: A sample Python script typed on the text editor Emacs.

```
% ls -l
total 1
-rw-r--r--  1 daisuke  taiwan  75 Feb 23 22:13 ao2021_s01_01.py
% cat ao2021_s01_01.py
#!/usr/pkg/bin/python3.9

print ("This is a very simple Python 3 script.")
```

5.4 Executing a Python script

For executing a Python script, try following. (Fig. 4)

```
% ls -lF
total 1
-rw-r--r--  1 daisuke  taiwan  75 Feb 23 22:14 ao2021_s01_01.py
% chmod a+x ao2021_s01_01.py
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan  75 Feb 23 22:14 ao2021_s01_01.py*
% ./ao2021_s01_01.py
This is a very simple Python 3 script.
```

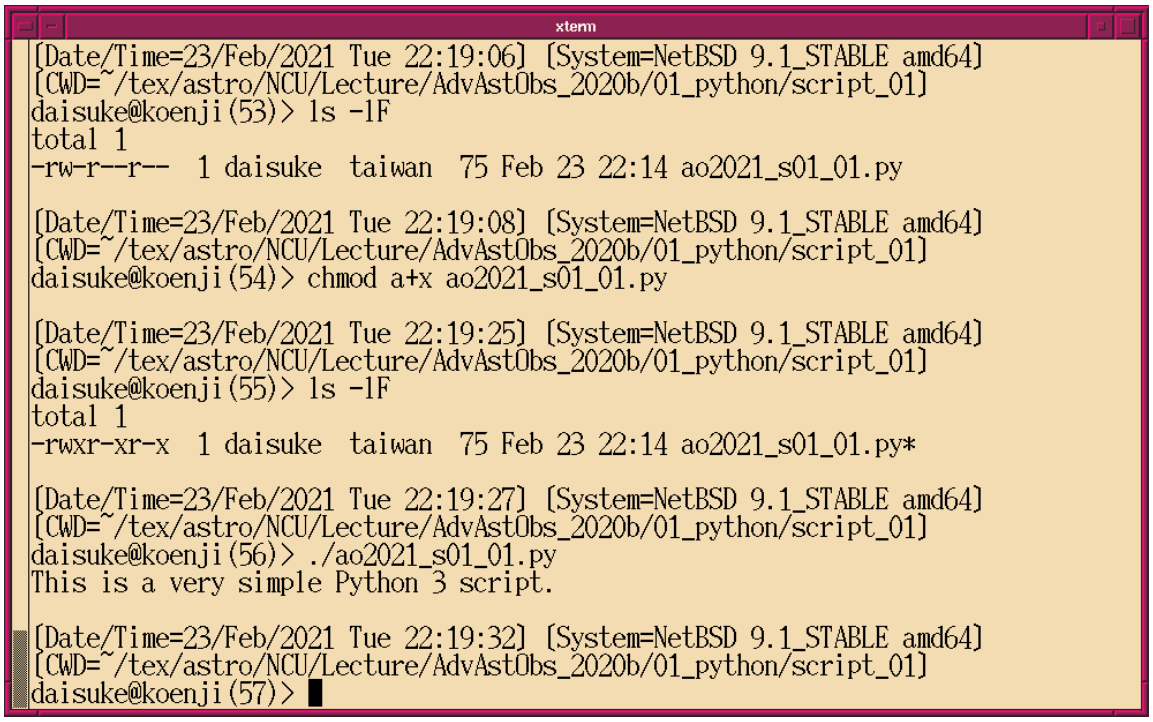

Figure 4: The way to execute a Python script on a terminal emulator.

6 Using argparse module

To handle command-line arguments, `argparse` module is extremely helpful. Try `argparse` module.

6.1 Adding two integers

Make a Python script to add two integers and return the result. Here is an example.

Python Code 2: ao2021_s01_02.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# construction of parser object
parser = argparse.ArgumentParser (description='Adding two numbers')

# adding arguments
parser.add_argument ('-a', type=int, default=1, help='number 1')
parser.add_argument ('-b', type=int, default=1, help='number 2')

# command-line argument analysis
args = parser.parse_args()

# two numbers
a = args.a
b = args.b

# adding two numbers
c = a + b

# printing result
print (a, '+', b, '=', c)
```

Execute the script.

```
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan   75 Feb 23 22:14 ao2021_s01_01.py*
-rw-r--r--  1 daisuke  taiwan  496 Feb 23 22:47 ao2021_s01_02.py
% chmod a+x ao2021_s01_02.py
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan   75 Feb 23 22:14 ao2021_s01_01.py*
-rwxr-xr-x  1 daisuke  taiwan  496 Feb 23 22:47 ao2021_s01_02.py*
% ./ao2021_s01_02.py -h
usage: ao2021_s01_02.py [-h] [-a A] [-b B]

Adding two numbers

optional arguments:
  -h, --help  show this help message and exit
  -a A        number 1
  -b B        number 2

% ./ao2021_s01_02.py -a 2 -b 3
2 + 3 = 5
% ./ao2021_s01_02.py -a 4 -b 7
4 + 7 = 11
% ./ao2021_s01_02.py -a 256 -b 768
256 + 768 = 1024
```

6.2 Adding two floating point numbers

If you specify a floating point number for ao2021_s01_02.py, you get an error.

```
% ./ao2021_s01_02.py -a 1.0 -b 2
usage: ao2021_s01_02.py [-h] [-a A] [-b B]
ao2021_s01_02.py: error: argument -a: invalid int value: '1.0'
```

To add two floating point numbers, try following.

Python Code 3: ao2021_s01_03.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# construction of parser object
desc = 'Adding two floating point numbers'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-a', type=float, default=1.0, help='number 1')
parser.add_argument ('-b', type=float, default=1.0, help='number 2')

# command-line argument analysis
args = parser.parse_args()

# two numbers
a = args.a
b = args.b

# adding two numbers
c = a + b

# printing result
print (a, '+', b, '=', c)
```

Execute the script.

```
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan   75 Feb 23 22:14 ao2021_s01_01.py*
-rwxr-xr-x  1 daisuke  taiwan  496 Feb 23 22:47 ao2021_s01_02.py*
-rw-r--r--  1 daisuke  taiwan  519 Feb 23 22:53 ao2021_s01_03.py
% chmod a+x ao2021_s01_03.py
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan   75 Feb 23 22:14 ao2021_s01_01.py*
-rwxr-xr-x  1 daisuke  taiwan  496 Feb 23 22:47 ao2021_s01_02.py*
-rwxr-xr-x  1 daisuke  taiwan  519 Feb 23 22:53 ao2021_s01_03.py*
% ./ao2021_s01_03.py -h
usage: ao2021_s01_03.py [-h] [-a A] [-b B]

Adding two floating point numbers

optional arguments:
  -h, --help  show this help message and exit
```

```
  -a A          number 1
  -b B          number 2

% ./ao2021_s01_03.py -a 1.234 -b 5.678
1.234 + 5.678 = 6.912
```

6.3 Arithmetic operations

Make a Python script to carry out arbitrary arithmetic operation of two numbers.

Python Code 4: ao2021_s01_04.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# construction of parser object
desc = 'Arithmetic operation of two numbers'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('n1', type=float, default=1.0, help='number 1')
parser.add_argument ('operator', choices=['+', '-', 'x', '/'], help='operator')
parser.add_argument ('n2', type=float, default=1.0, help='number 2')

# command-line argument analysis
args = parser.parse_args()

# two numbers
n1 = args.n1
operator = args.operator
n2 = args.n2

# calculation
if (operator == '+'):
    n3 = n1 + n2
elif (operator == '-'):
    n3 = n1 - n2
elif (operator == 'x'):
    n3 = n1 * n2
elif (operator == '/'):
    n3 = n1 / n2

# printing result
print (n1, operator, n2, '=', n3)
```

Try the script.

```
% ls -lF
total 1
-rwxr-xr-x  1 daisuke  taiwan   75 Feb 23 22:14 ao2021_s01_01.py*
-rwxr-xr-x  1 daisuke  taiwan  496 Feb 23 22:47 ao2021_s01_02.py*
-rwxr-xr-x  1 daisuke  taiwan  519 Feb 23 22:53 ao2021_s01_03.py*
-rw-r--r--  1 daisuke  taiwan  783 Feb 23 23:02 ao2021_s01_04.py
% chmod a+x ao2021_s01_04.py
% ls -lF
total 1
```

```
-rwxr-xr-x  1 daisuke  taiwan   75 Feb 23 22:14 ao2021_s01_01.py*
-rwxr-xr-x  1 daisuke  taiwan  496 Feb 23 22:47 ao2021_s01_02.py*
-rwxr-xr-x  1 daisuke  taiwan  519 Feb 23 22:53 ao2021_s01_03.py*
-rwxr-xr-x  1 daisuke  taiwan  783 Feb 23 23:02 ao2021_s01_04.py*
% ./ao2021_s01_04.py -h
usage: ao2021_s01_04.py [-h] n1 {+,-,x,/} n2

Arithmetic operation of two numbers

positional arguments:
  n1           number 1
  {+,-,x,/}    operator
  n2           number 2

optional arguments:
  -h, --help   show this help message and exit

% ./ao2021_s01_04.py 1.2 + 3.4
1.2 + 3.4 = 4.6
% ./ao2021_s01_04.py 9.8 - 7.6
9.8 - 7.6 = 2.200000000000001
% ./ao2021_s01_04.py 12.3 x 45.6
12.3 x 45.6 = 560.88
% ./ao2021_s01_04.py 123.45 / 6.78
123.45 / 6.78 = 18.207964601769913
```

6.4 Calculation of a mean

Make a Python script to calculate a mean for a given set of numbers.

Python Code 5: ao2021_s01_05.py

```python
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# construction of parser object
desc = 'Calculation of a mean'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('numbers', type=float, nargs='+', help='a set of numbers')

# command-line argument analysis
args = parser.parse_args()

# numbers
numbers = args.numbers

# number of data
n = len (numbers)
# initialisation of a variable 'total'
total = 0.0
# adding numbers
for number in numbers:
    total += number
# calculation of a mean
mean = total / n
```

```
# printing result
print ('input data     =', numbers)
print ('number of data =', n)
print ('mean           =', mean)
```

Execute the script.

```
% chmod a+x ao2021_s01_05.py
% ./ao2021_s01_05.py -h
usage: ao2021_s01_05.py [-h] numbers [numbers ...]

Calculation of a mean

positional arguments:
  numbers     a set of numbers

optional arguments:
  -h, --help  show this help message and exit

% ./ao2021_s01_05.py 10.0 9.0 11.0 8.0 12.0 13.0 7.0
input data     = [10.0, 9.0, 11.0, 8.0, 12.0, 13.0, 7.0]
number of data = 7
mean           = 10.0
% ./ao2021_s01_05.py 1 2 3 4 5 6 7 8 9 10
input data     = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
number of data = 10
mean           = 5.5
```

Modify above script to allow the calculation of a median.

Python Code 6: ao2021_s01_06.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# construction of parser object
desc = 'Calculation of average'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-a', choices=['mean', 'median'], \
                     help='algorithm (mean or median)')
parser.add_argument ('numbers', type=float, nargs='+', help='a set of numbers')

# command-line argument analysis
args = parser.parse_args()

# numbers
algorithm = args.a
numbers = args.numbers

# number of data
n = len (numbers)

# calculation of average
```

```
if (algorithm == 'mean'):
    # initialisation of a variable 'total'
    total = 0.0
    # adding numbers
    for number in numbers:
        total += number
    # calculation of a mean
    mean = total / n
elif (algorithm == 'median'):
    # sorting data
    sorted_numbers = sorted (numbers)
    # calculation of a median
    if (n % 2 == 0):
        median = ( sorted_numbers[int (n / 2) - 1] \
                 + sorted_numbers[int (n / 2)] ) / 2.0
    elif (n % 2 == 1):
        median = sorted_numbers[int (n / 2)]

# printing result
print ('input data     =', numbers)
print ('number of data =', n)
if (algorithm == 'mean'):
    print ('mean           =', mean)
elif (algorithm == 'median'):
    print ('median         =', median)
```

Try the script.

```
% chmod a+x ao2021_s01_06.py
% ./ao2021_s01_06.py -h
usage: ao2021_s01_06.py [-h] [-a {mean,median}] numbers [numbers ...]

Calculation of average

positional arguments:
  numbers            a set of numbers

optional arguments:
  -h, --help         show this help message and exit
  -a {mean,median}   algorithm (mean or median)

% ./ao2021_s01_06.py -a mean 10 11 12 13 14 15 16 17 18 19
input data     = [10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0]
number of data = 10
mean           = 14.5
% ./ao2021_s01_06.py -a median 10 11 12 13 14 15 16 17 18 19
input data     = [10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0]
number of data = 10
median         = 14.5
% ./ao2021_s01_06.py -a mean 10.0 10.1 9.9 10.2 9.8 10.3 9.7 30.0
input data     = [10.0, 10.1, 9.9, 10.2, 9.8, 10.3, 9.7, 30.0]
number of data = 8
mean           = 12.5
% ./ao2021_s01_06.py -a median 10.0 10.1 9.9 10.2 9.8 10.3 9.7 30.0
input data     = [10.0, 10.1, 9.9, 10.2, 9.8, 10.3, 9.7, 30.0]
number of data = 8
median         = 10.05
```

6.5 Finding prime numbers

Make a Python script to find prime numbers and write those numbers into a file.

Python Code 7: ao2021_s01_07.py

```
#!/usr/pkg/bin/python3.9

# importing argparse module
import argparse

# importing pathlib module
import pathlib

# construction of parser object
desc = 'Finding prime numbers'
parser = argparse.ArgumentParser (description=desc)

# adding arguments
parser.add_argument ('-o', '--output', default='primenumbers.data', \
                     help='output file name')
parser.add_argument ('-s', '--n1', type=int, default=2, \
                     help='number to start')
parser.add_argument ('-e', '--n2', type=int, default=100, \
                     help='number to end')

# command-line argument analysis
args = parser.parse_args()

# parameters
file_output = args.output
n_start     = args.n1
n_end       = args.n2

# initialisation of a list to store results
primenumbers = []

# checking numbers from n_start to n_end
for i in range (n_start, n_end + 1):
    # resetting the parameter "count"
    count = 0
    # examining if the number is divisible by numbers between 2 and (i-1)
    for j in range (2, i):
        # if the number is divisible, then adding 1 to "count"
        if (i % j == 0):
            count += 1
            # if count is >= 1, the number is not a prime number
            break
    # if the number is a prime number, add it to the list "primenumbers"
    if (count == 0):
        primenumbers.append (i)

# making a string for output
string_primenumbers = ''
for i in primenumbers:
    string_primenumbers += ("%d\n" % i)

# writing result into a file
path_output = pathlib.Path (file_output)
path_output.write_text (string_primenumbers)
```

Try the script.

```
% chmod a+x ao2021_s01_07.py
% ./ao2021_s01_07.py -h
usage: ao2021_s01_07.py [-h] [-o OUTPUT] [-s N1] [-e N2]

Finding prime numbers

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        output file name
  -s N1, --n1 N1        number to start
  -e N2, --n2 N2        number to end

% ./ao2021_s01_07.py -o prime_10.data -s 2 -e 10
% ls
ao2021_s01_01.py*   ao2021_s01_03.py~*  ao2021_s01_05.py~   ao2021_s01_07.py*
ao2021_s01_02.py*   ao2021_s01_04.py*   ao2021_s01_06.py*   ao2021_s01_07.py~*
ao2021_s01_03.py*   ao2021_s01_05.py*   ao2021_s01_06.py~   prime_10.data
% cat prime_10.data
2
3
5
7
% ./ao2021_s01_07.py --output prime_100.data --n1 2 --n2 100
% ls
ao2021_s01_01.py*   ao2021_s01_04.py*   ao2021_s01_06.py~   prime_100.data
ao2021_s01_02.py*   ao2021_s01_05.py*   ao2021_s01_07.py*
ao2021_s01_03.py*   ao2021_s01_05.py~   ao2021_s01_07.py~*
ao2021_s01_03.py~*  ao2021_s01_06.py*   prime_10.data
% tail prime_100.data
53
59
61
67
71
73
79
83
89
97
% ./ao2021_s01_07.py --output prime_10000.data --n1 2 --n2 10000
% tail prime_10000.data
9887
9901
9907
9923
9929
9931
9941
9949
9967
9973
```

7 For your training

- If you do not have Linux (or BSD) on your computer, install Linux (or BSD) on your computer. Describe the way to install Linux (or BSD).
- If you do not have Python on your computer, install the latest version of Python on your computer. Describe the way to install Python.
- If you do not have NumPy, SciPy, Matplotlib, Astropy, and Astropy affiliated packages, install them on your computer. Describe the way to install those packages.
- Find your favourite text editor. Describe how you like it. What is the unique feature of your favourite text editor?
- Make three Python scripts which use `argparse` module. Show the source code of those Python script. Execute them. Take a screenshot of your computer display, and show the result of the execution.
- Think about a useful Python script utilising `argparse` module. Describe the design of it.
- Read "Python Tutorial" on the Python official website, and learn more about Python.
- The course "Astroinformatics" offers an opportunity to learn more about basic Python programming. If you are new to Python, think about taking the course "Astroinformatics".

8 Assignment

There is no assignment for this session.