Astroinformatics 2022
Session 02: Using Useful Python Modules

Kinoshita Daisuke

19 September 2022
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 "Astroinformatics" (course ID: AS6095) offered at Institute of Astronomy, National Central University from September 2022 to January 2023.
  - 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/`

For this session, we try useful Python modules.

1 Sample Python scripts for this session

Sample Python scripts for this session can be downloaded from GitHub repository. Visit following GitHub repository.

- `https://github.com/kinoshitadaisuke/ncu_astroinformatics_202209`

1.1 Executing sample Python scripts on a terminal emulator

If you prefer to execute sample Python scripts for this session on a terminal emulator, download `.py` files from GitHub repository.

1.2 Executing sample Python scripts on JupyterLab

If you prefer to execute sample Python scripts for this session on JupyterLab (or Jupyter Notebook), download `.ipynb` file from GitHub repository.

1.3 Executing sample Python scripts using Binder

If you prefer to execute sample Python scripts for this session on Binder, visit following web page.

- `https://mybinder.org/v2/gh/kinoshitadaisuke/ncu_astroinformatics_202209/HEAD`

Start your favourite web browser and go to above web page. (Fig. 1) In a minute or two, you see JupyterLab working on your web browser. (Fig. 2) Go to the directory (folder) "`s02`". (Fig. 3) Choose the file "`ai202209_s02.ipynb`" (Fig. 4 and 5) and open it (Fig. 6).

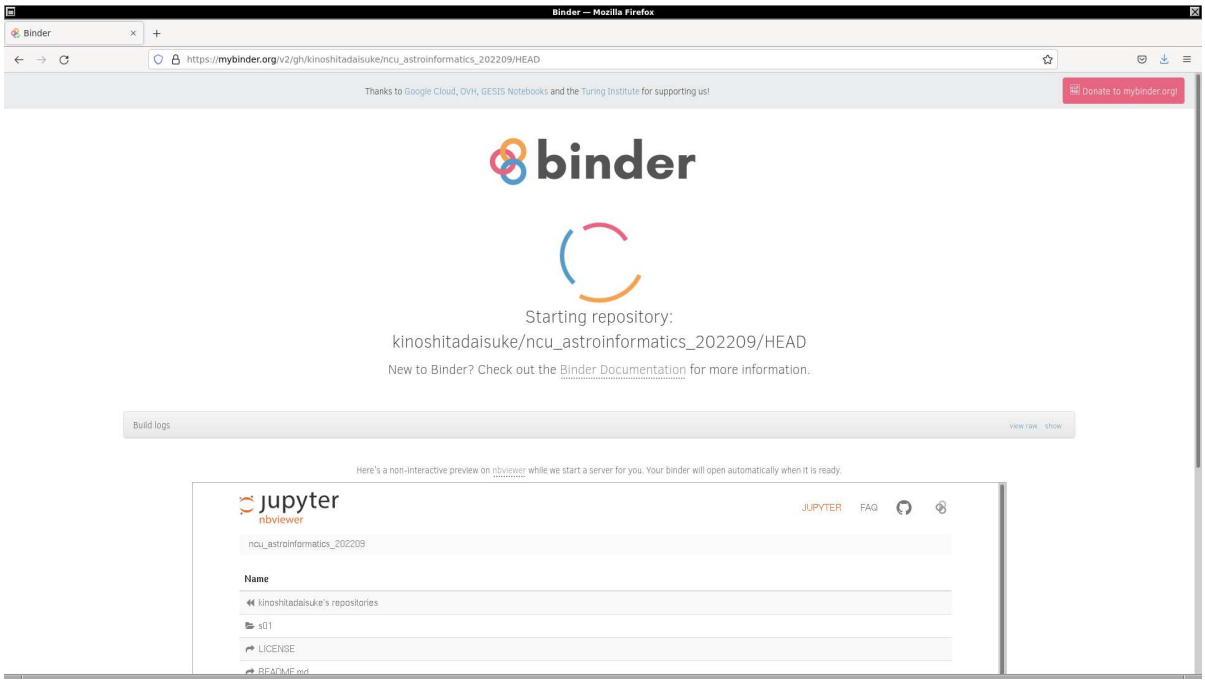

Figure 1: Using Binder to execute sample Python scripts for this session.

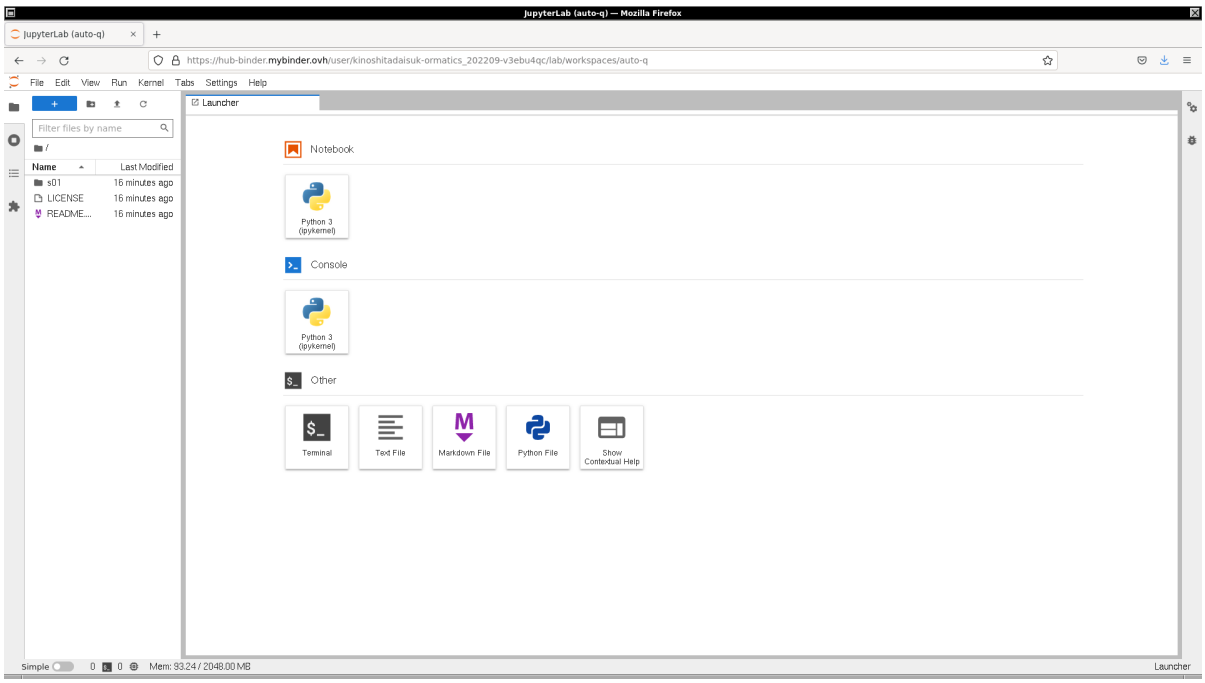

Figure 2: Using Binder to execute sample Python scripts for this session.

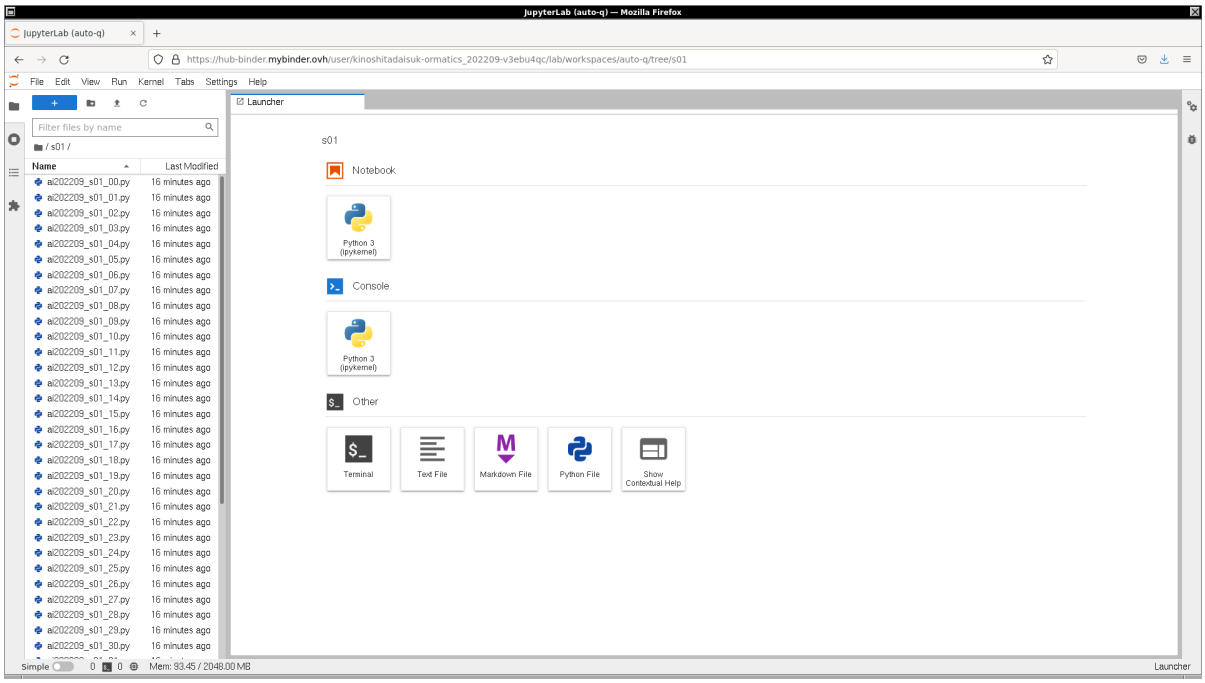

Figure 3: Using Binder to execute sample Python scripts for this session.

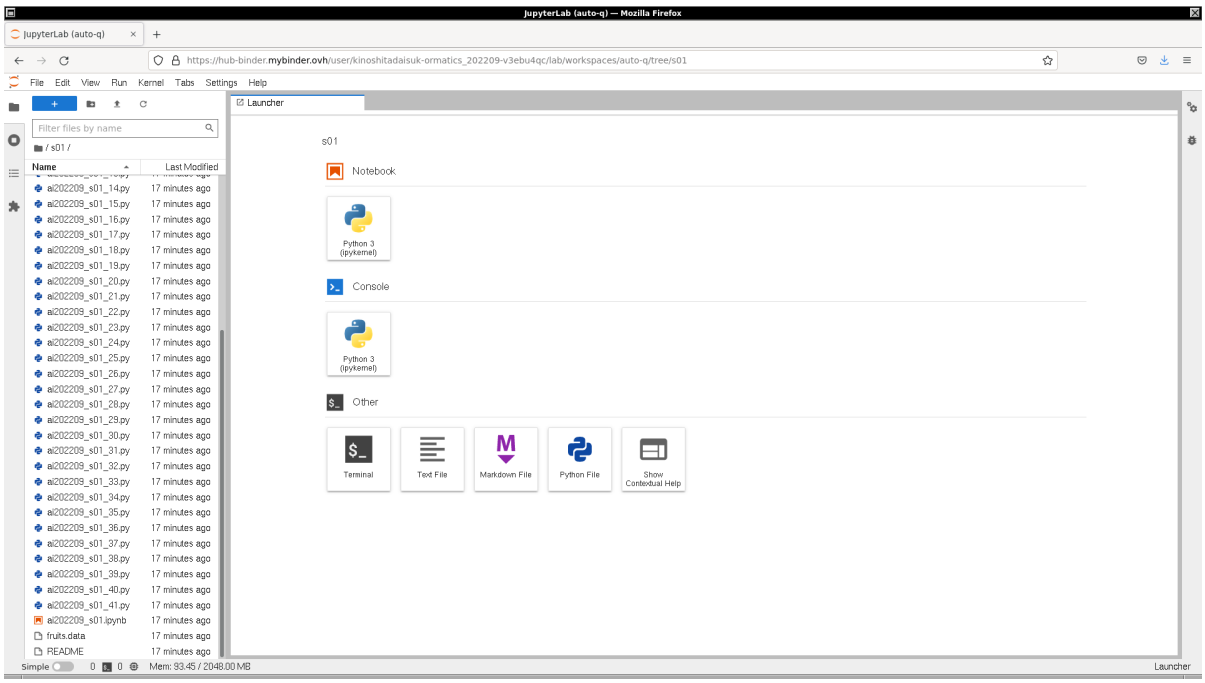

Figure 4: Using Binder to execute sample Python scripts for this session.

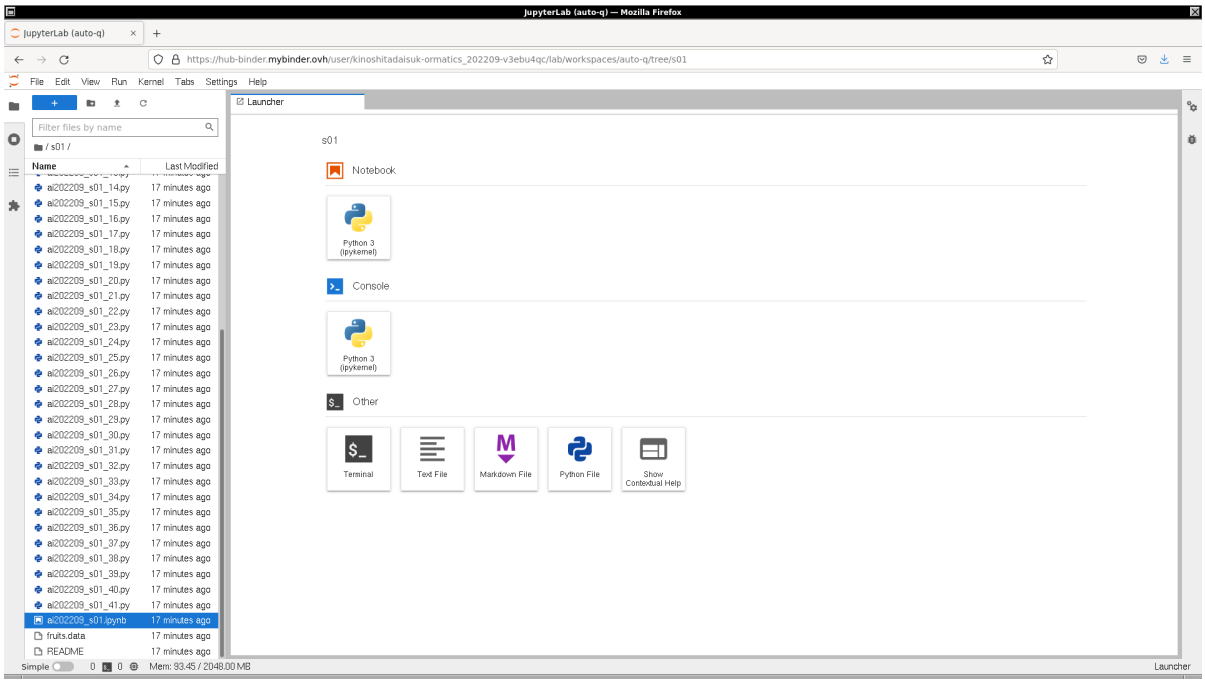

Figure 5: Using Binder to execute sample Python scripts for this session.

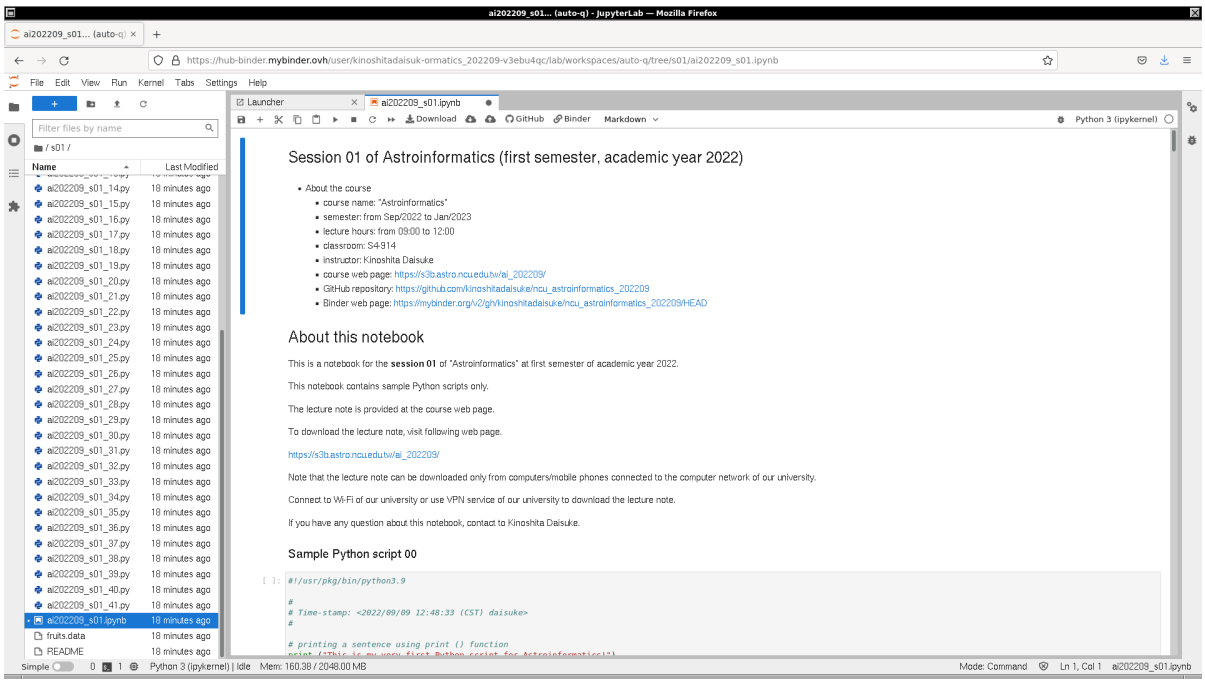

Figure 6: Using Binder to execute sample Python scripts for this session.

2 Using `math` module

Try `math` module.

2.1 Constants

We can get some constants using `math` module. Try following script.

Python Code 1: ai202209_s02_00.py

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

#
# Time-stamp: <2022/09/16 16:08:59 (CST) daisuke>
#

# importing math module
import math

#
# some constants
#

# pi
pi = math.pi
print (f'pi    = {pi}')

# tau
tau = math.tau
print (f'tau   = 2.0 * pi\n      = {tau}')

# e
e = math.e
print (f'e     = {e}')

# positive infinity
pinf = math.inf
print (f'+inf = {pinf}')

# negative infinity
ninf = -math.inf
print (f'-inf = {ninf}')

# NaN (not a number)
nan = math.nan
print (f'NaN   = {nan}')
```

```
% ./ai202209_s02_00.py
pi    = 3.141592653589793
tau   = 2.0 * pi
      = 6.283185307179586
e     = 2.718281828459045
+inf = inf
-inf = -inf
NaN   = nan
```

Try following practice.

Practice 02-01

Make your own Python script to calculate e^5 and print the result of your calculation.

2.2 Useful functions

Useful functions, such as `floor ()`, `ceil ()`, `fabs ()`, are provided by `math` module. Try those functions. Here is an example.

Python Code 2: ai202209_s02_01.py

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

#
# Time-stamp: <2022/09/16 16:09:04 (CST) daisuke>
#

# importing math module
import math

#
# some useful functions
#

# two floats "a" and "b"
a = 12.34
b = -56.78

# floor
a_floor = math.floor (a)
b_floor = math.floor (b)

# printing results
print (f'Use of floor () function:')
print (f'  a          = {a}')
print (f'  floor (a) = {a_floor}')
print (f'  {a_floor} is the largest integer less than or equal to {a}.')
print (f'  b          = {b}')
print (f'  floor (b) = {b_floor}')
print (f'  {b_floor} is the largest integer less than or equal to {b}.')
print (f'')

# ceil
a_ceil = math.ceil (a)
b_ceil = math.ceil (b)

# printing results
print (f'Use of ceil () function:')
print (f'  a          = {a}')
print (f'  ceil (a)   = {a_ceil}')
print (f'  {a_ceil} is the smallest integer greater than or equal to {a}.')
print (f'  b          = {b}')
print (f'  ceil (b) = {b_ceil}')
print (f'  {b_ceil} is the smallest integer greater than or equal to {b}.')
print (f'')

# trunc
a_trunc = math.trunc (a)
b_trunc = math.trunc (b)
```

```
# printing results
print (f'Use of trunc () function:')
print (f'  a          = {a}')
print (f'  trunc (a)  = {a_trunc}')
print (f'  {a_trunc} is the integer part of {a}.')
print (f'  b          = {b}')
print (f'  trunc (b) = {b_trunc}')
print (f'  {b_trunc} is the integer part of {b}.')
print (f'')

# modf
(a_fractional, a_integer) = math.modf (a)
(b_fractional, b_integer) = math.modf (b)

# printing results
print (f'Use of modf () function:')
print (f'  a                     = {a}')
print (f'  integer part of a     = {a_integer}')
print (f'  fractional part of a = {a_fractional}')
print (f'  b                     = {b}')
print (f'  integer part of b     = {b_integer}')
print (f'  fractional part of b = {b_fractional}')
print (f'')

# fabs
a_abs = math.fabs (a)
b_abs = math.fabs (b)

# printing results
print (f'Use of fabs () function:')
print (f'  a                    = {a}')
print (f'  absolute value of a = {a_abs}')
print (f'  b                    = {b}')
print (f'  absolute value of b = {b_abs}')
print (f'')

# two integers "c" and "d"
c = 30
d = 45

# gcd
gcd_c_d = math.gcd (c, d)

# printing results
print (f'Use of gcd () function:')
print (f'  c                                    = {c}')
print (f'  d                                    = {d}')
print (f'  greatest common divisor of c and d = {gcd_c_d}')
print (f'')

# lcm
lcm_c_d = math.lcm (c, d)

# printing results
print (f'Use of lcm () function:')
print (f'  c                                  = {c}')
print (f'  d                                  = {d}')
print (f'  least common multiple of c and d = {lcm_c_d}')
print (f'')
```

```
# factorial
factorial_5 = math.factorial (5)
factorial_8 = math.factorial (8)

# printing results
print (f'Use of factorial () function:')
print (f'  5! = 1 * 2 * 3 * 4 * 5\n     = {factorial_5}')
print (f'  8! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8\n     = {factorial_8}')
print (f'')

# perm
p_5_3 = math.perm (5, 3)
p_8_4 = math.perm (8, 4)

fac_2 = math.factorial (2)
fac_4 = math.factorial (4)
fac_5 = math.factorial (5)
fac_8 = math.factorial (8)

# printing results
print (f'Use of perm () function:')
print (f'  P (5, 3) = (5!) / (2!)')
print (f'           = {fac_5} / {fac_2}')
print (f'           = {p_5_3}')
print (f'  P (8, 4) = (8!) / (4!)')
print (f'           = {fac_8} / {fac_4}')
print (f'           = {p_8_4}')
print (f'')

# comb
c_5_3 = math.comb (5, 3)
c_8_4 = math.comb (8, 4)

fac_2 = math.factorial (2)
fac_3 = math.factorial (3)
fac_4 = math.factorial (4)
fac_5 = math.factorial (5)
fac_8 = math.factorial (8)

# printing results
print (f'Use of comb () function:')
print (f'  C (5, 3) = (5!) / (3! * 2!)')
print (f'           = {fac_5} / ({fac_3} * {fac_2})')
print (f'           = {c_5_3}')
print (f'  C (8, 4) = (8!) / (4! * 4!)')
print (f'           = {fac_8} / ({fac_4} * {fac_4})')
print (f'           = {c_8_4}')
print (f'')
```

```
% ./ai202209_s02_01.py
Use of floor () function:
  a         = 12.34
  floor (a) = 12
  12 is the largest integer less than or equal to 12.34.
  b         = -56.78
  floor (b) = -57
  -57 is the largest integer less than or equal to -56.78.
```

```
Use of ceil () function:
  a         = 12.34
  ceil (a)  = 13
  13 is the smallest integer greater than or equal to 12.34.
  b         = -56.78
  ceil (b) = -56
  -56 is the smallest integer greater than or equal to -56.78.

Use of trunc () function:
  a         = 12.34
  trunc (a)  = 12
  12 is the integer part of 12.34.
  b         = -56.78
  trunc (b) = -56
  -56 is the integer part of -56.78.

Use of modf () function:
  a                    = 12.34
  integer part of a    = 12.0
  fractional part of a = 0.33999999999999986
  b                    = -56.78
  integer part of b    = -56.0
  fractional part of b = -0.7800000000000011

Use of fabs () function:
  a                   = 12.34
  absolute value of a = 12.34
  b                   = -56.78
  absolute value of b = 56.78

Use of gcd () function:
  c                                  = 30
  d                                  = 45
  greatest common divisor of c and d = 15

Use of lcm () function:
  c                                = 30
  d                                = 45
  least common multiple of c and d = 90

Use of factorial () function:
  5! = 1 * 2 * 3 * 4 * 5
     = 120
  8! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8
     = 40320

Use of perm () function:
  P (5, 3) = (5!) / (2!)
           = 120 / 2
           = 60
  P (8, 4) = (8!) / (4!)
           = 40320 / 24
           = 1680

Use of comb () function:
  C (5, 3) = (5!) / (3! * 2!)
           = 120 / (6 * 2)
           = 10
```

```
C (8, 4) = (8!) / (4! * 4!)
         = 40320 / (24 * 24)
         = 70
```

Try following practice.

Practice 02-02

Make your own Python script to use `fabs ()` function to obtain the absolute value of a given number, and print the result of your calculation.

2.3 Mathematical functions

Mathematical functions, such as `log ()`, `exp ()`, `sin ()`, are provided by `math` module. Try those functions. Here is an example.

Python Code 3: ai202209_s02_02.py

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

#
# Time-stamp: <2022/09/16 16:09:09 (CST) daisuke>
#

# importing math module
import math

#
# some mathematical functions
#

# pow () function
a = math.pow (3, 4)

# printing result
print (f'3^4 = 3 * 3 * 3 * 3 = {a}')
print (f'')

# sqrt () function
b = math.sqrt (3)

# printing result
print (f'sqrt (3) = {b}')
print (f'')

# log () function
c = math.log (100)

# printing result
print (f'log (100) = {c}')
print (f'')

# log () function
d = math.log10 (100)

# printing result
print (f'log10 (100) = {d}')
print (f'')
```

```
# conversion from degree into radian
e_deg = 180.0
e_rad = math.radians (e_deg)

# printing result
print (f'{e_deg} deg = {e_rad} rad')
print (f'')

# conversion from radian into degree
f_rad = math.pi / 2.0
f_deg = math.degrees (f_rad)

# printing result
print (f'{f_rad} rad = {f_deg} deg')
print (f'')

# sin (), cos (), and tan ()
g_deg = 60.0
g_rad = math.radians (g_deg)
sin_g = math.sin (g_rad)
cos_g = math.cos (g_rad)
tan_g = math.tan (g_rad)

# printing result
print (f'{g_deg} deg       = {g_rad} rad')
print (f'sin ({g_deg} deg) = {sin_g}')
print (f'cos ({g_deg} deg) = {cos_g}')
print (f'tan ({g_deg} deg) = {tan_g}')
print (f'')

# distance between two points
coord_0  = (0.0, 1.0)
coord_1  = (3.0, 2.0)
dist_0_1 = math.dist (coord_0, coord_1)
coord_2  = (0.0, 0.0, 0.0)
coord_3  = (1.0, 1.0, 1.0)
dist_2_3 = math.dist (coord_2, coord_3)

# printing result
print (f'coord_0 = {coord_0}')
print (f'coord_1 = {coord_1}')
print (f'distance between coord_0 and coord_1 = {dist_0_1}')
print (f'coord_2 = {coord_2}')
print (f'coord_3 = {coord_3}')
print (f'distance between coord_2 and coord_3 = {dist_2_3}')
print (f'')
```

```
% ./ai202209_s02_02.py
3^4 = 3 * 3 * 3 * 3 = 81.0

sqrt (3) = 1.7320508075688772

log (100) = 4.605170185988092

log10 (100) = 2.0

180.0 deg = 3.141592653589793 rad
```

```
1.5707963267948966 rad = 90.0 deg

60.0 deg       = 1.0471975511965976 rad
sin (60.0 deg) = 0.8660254037844386
cos (60.0 deg) = 0.5000000000000001
tan (60.0 deg) = 1.7320508075688767

coord_0 = (0.0, 1.0)
coord_1 = (3.0, 2.0)
distance between coord_0 and coord_1 = 3.1622776601683795
coord_2 = (0.0, 0.0, 0.0)
coord_3 = (1.0, 1.0, 1.0)
distance between coord_2 and coord_3 = 1.7320508075688772
```

Try following practice.

Practice 02-03

Make your own Python script to calculate the distance between $X = (1.0, 2.0, 3.0)$ and $Y = (4.0, 5.0, 6.0)$ and print the result of your calculation.

3 Using argparse module

Command-line arguments analysis can be carried out by using `argparse` module. It is possible to make a versatile and flexible Python program with `argparse` module. It is actually a very powerful tool for developing a data analysis pipeline and a script for model calculations.

3.1 An example of using argparse module

Here is an example of using `argparse` module.

Python Code 4: ai202209_s02_03.py

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

#
# Time-stamp: <2022/09/17 18:46:17 (CST) daisuke>
#

# importing argparse module
import argparse

# command-line argument analysis
parser = argparse.ArgumentParser (description='adding two integers')
parser.add_argument ('-a', type=int, default=1, help='number 1')
parser.add_argument ('-b', type=int, default=1, help='number 2')
args = parser.parse_args ()

# input parameters
a = args.a
b = args.b

# calculation
c = a + b

# printing result
print (f'{a} + {b} = {c}')
```

```
% ./ai202209_s02_03.py -h
usage: ai202209_s02_03.py [-h] [-a A] [-b B]

adding two integers

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

% ./ai202209_s02_03.py -a 3 -b 5
3 + 5 = 8
% ./ai202209_s02_03.py -a 100 -b 200
100 + 200 = 300
```

Try following practice.

Practice 02-04

Make your own Python script to calculate a multiplication of arbitrary two numbers and print the result of your calculation.

3.2 One more example of using argparse module

Here is one more example of using `argparse` module.

Python Code 5: ai202209_s02_04.py

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

#
# Time-stamp: <2022/09/17 18:58:14 (CST) daisuke>
#

# importing argparse module
import argparse

# list of available operators
list_operators = ['+', '-', 'x', '/']

# command-line arguments analysis
parser = argparse.ArgumentParser (description='arithmetic calculations')
parser.add_argument ('number1', type=float, help='number1')
parser.add_argument ('operator', choices=list_operators, help='operator')
parser.add_argument ('number2', type=float, help='number2')
args = parser.parse_args ()

# input parameters
n1 = args.number1
n2 = args.number2
op = args.operator

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

```
elif (op == '/'):
    n3 = n1 / n2

# printing result
print (f'{n1} {op} {n2} = {n3}')
```

```
% ./ai202209_s02_04.py -h
usage: ai202209_s02_04.py [-h] number1 {+,-,x,/} number2

arithmetic calculations

positional arguments:
  number1     number1
  {+,-,x,/}   operator
  number2     number2

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

% ./ai202209_s02_04.py 2 + 3
2.0 + 3.0 = 5.0
% ./ai202209_s02_04.py 7 - 4
7.0 - 4.0 = 3.0
% ./ai202209_s02_04.py 5 x 8
5.0 x 8.0 = 40.0
% ./ai202209_s02_04.py 9 / 6
9.0 / 6.0 = 1.5
```

Try following practice.

Practice 02-05

Make a convenient calculator using Python.

4 Using os module

The `os` module provides interfaces to the operating system. Try following example.

Python Code 6: ai202209_s02_05.py

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

#
# Time-stamp: <2022/09/17 21:00:23 (CST) daisuke>
#

# importing os module
import os

#
# printing current working directory
#

# knowing where you are now
cwd = os.getcwd ()

# printing where you are now
print (f'current working directory = {cwd}')
```

```
#
# printing files at current working directory
#

# getting a list of files at current working directory
list_files = os.listdir ()

# printing list of files at current working directory
print (f'list of files at current working directory:')
for file in sorted (list_files):
    print (f'  {file}')

#
# the other way to print files at current working directory
#

# the other way to get a list of files at current working directory
list_files2 = os.scandir ()

# printing list of files at current working directory
print (f'list of files at current working directory:')
for file in sorted (list_files2, key=lambda x: x.name):
    print (f'  {file.name}')

#
# printing information about a file
#

# getting information about a file
filename = 'ai202209_s02_00.py'
statinfo = os.stat (filename)

# printing size of file
print (f'information of file "{filename}":')
print (f'  size : {statinfo.st_size} byte')
```

```
% ./ai202209_s02_05.py
current working directory = /tmp/test
list of files at current working directory:
  ai202209_s02_00.py
  ai202209_s02_01.py
  ai202209_s02_02.py
  ai202209_s02_03.py
  ai202209_s02_04.py
  ai202209_s02_05.py
  ai202209_s02_06.py
  ai202209_s02_07.py
  ai202209_s02_08.py
  ai202209_s02_09.py
  ai202209_s02_10.py
  ai202209_s02_11.py
  ai202209_s02_12.py
  ai202209_s02_13.py
  ai202209_s02_14.py
  ai202209_s02_15.py
  ai202209_s02_16.py
  ai202209_s02_17.py
```

```
  ai202209_s02_18.py
  ai202209_s02_19.py
  ai202209_s02_20.py
  ai202209_s02_21.py
  ai202209_s02_22.py
  ai202209_s02_23.py
  ai202209_s02_24.py
  ai202209_s02_25.py
  ai202209_s02_26.py
  ai202209_s02_27.py
  ai202209_s02_28.py
  ai202209_s02_29.py
  ai202209_s02_30.py
  ai202209_s02_31.py
  ai202209_s02_32.py
  ai202209_s02_33.py
  ai202209_s02_34.py
  ai202209_s02_35.py
  bsc.data
  catalog.gz
  pi_1000.txt
  pi_1000_2.txt
list of files at current working directory:
  ai202209_s02_00.py
  ai202209_s02_01.py
  ai202209_s02_02.py
  ai202209_s02_03.py
  ai202209_s02_04.py
  ai202209_s02_05.py
  ai202209_s02_06.py
  ai202209_s02_07.py
  ai202209_s02_08.py
  ai202209_s02_09.py
  ai202209_s02_10.py
  ai202209_s02_11.py
  ai202209_s02_12.py
  ai202209_s02_13.py
  ai202209_s02_14.py
  ai202209_s02_15.py
  ai202209_s02_16.py
  ai202209_s02_17.py
  ai202209_s02_18.py
  ai202209_s02_19.py
  ai202209_s02_20.py
  ai202209_s02_21.py
  ai202209_s02_22.py
  ai202209_s02_23.py
  ai202209_s02_24.py
  ai202209_s02_25.py
  ai202209_s02_26.py
  ai202209_s02_27.py
  ai202209_s02_28.py
  ai202209_s02_29.py
  ai202209_s02_30.py
  ai202209_s02_31.py
  ai202209_s02_32.py
  ai202209_s02_33.py
  ai202209_s02_34.py
  ai202209_s02_35.py
```

```
  bsc.data
  catalog.gz
  pi_1000.txt
  pi_1000_2.txt
information of file "ai202209_s02_00.py":
  size : 468 byte
```

Try following practice.

Practice 02-06

Make your own Python script to find the location of your current working directory and print it.

5 Using sys module

The `sys` module provides functions for system specific parameters. Try following example.

Python Code 7: ai202209_s02_06.py

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

#
# Time-stamp: <2022/09/18 08:17:26 (CST) daisuke>
#

# importing sys module
import sys

#
# command-line arguments
#

# receiving command-line arguments
args = sys.argv

# printing command-line arguments
print (f'Command-line arguments:')
for i in range (len (args)):
    print (f'  sys.argv[{i}] = "{sys.argv[i]}"')

#
# platform information
#

# getting platform information
platform = sys.platform

# printing platform information
print (f'Platform information:')
print (f'  platform = {platform}')

#
# byte order of the system (big-endian? or little-endian?)
#

# finding byte order of the system
byteorder = sys.byteorder

# printing byte order of the system
```

```
print (f'Byte order:')
print (f'  byte order = {byteorder}')

#
# implementation of Python interpreter
#

# finding implementation of Python interpreter
implementation = sys.implementation

# printing implementation of Python interpreter
print (f'Implementation of Python interperter:')
print (f'  name : {implementation.name}')

#
# version of Python interpreter
#

# getting the version of Python interpreter
version = sys.version_info

# printing the version of Python interpreter
print (f'Version of Python interpreter:')
print (f'  version = {version[0]}.{version[1]}.{version[2]}')

#
# location of Python interpreter
#

# finding the absolute path of Python interpreter
location_python = sys.executable

# printing location of Python interpreter
print (f'Location of Python interpreter:')
print (f'  Python interpreter = {location_python}')

#
# list of already loaded modules
#

# getting a list of already loaded modules
list_modules = sys.modules

# printing a list of already loaded modules
print (f'List of already loaded modules:')
for module in sorted (list_modules):
    print (f'  {module}')

#
# exiting Python interpreter
#

# exit
sys.exit (1)
```

```
% ./ai202209_s02_06.py
Command-line arguments:
  sys.argv[0] = "./ai202209_s02_06.py"
```

```
Platform information:
  platform = netbsd9
Byte order:
  byte order = little
Implementation of Python interperter:
  name : cpython
Version of Python interpreter:
  version = 3.9.13
Location of Python interpreter:
  Python interpreter = /usr/pkg/bin/python3.9
List of already loaded modules:
  __main__
  _abc
  _bootlocale
  _codecs
  _collections
  _collections_abc
  _distutils_hack
  _frozen_importlib
  _frozen_importlib_external
  _functools
  _heapq
  _imp
  _io
  _locale
  _operator
  _signal
  _sitebuiltins
  _sre
  _stat
  _thread
  _warnings
  _weakref
  abc
  builtins
  codecs
  collections
  collections.abc
  contextlib
  copyreg
  encodings
  encodings.aliases
  encodings.latin_1
  encodings.utf_8
  enum
  functools
  genericpath
  heapq
  importlib
  importlib._bootstrap
  importlib._bootstrap_external
  importlib.abc
  importlib.machinery
  importlib.util
  io
  itertools
  keyword
  marshal
  mpl_toolkits
```

```
  operator
  os
  os.path
  posix
  posixpath
  re
  reprlib
  site
  sphinxcontrib
  sre_compile
  sre_constants
  sre_parse
  stat
  sys
  time
  types
  typing
  typing.io
  typing.re
  warnings
  zipimport
```

Try following practice.

Practice 02-07

Make your own Python script to find out the platform name and byte order of your computer and print those information.

6 Using `random` module

The `random` module can be used to generate random numbers. Random numbers are very useful for scientific calculations. Try following example.

Python Code 8: ai202209_s02_07.py

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

#
# Time-stamp: <2022/09/18 11:06:30 (CST) daisuke>
#

# importing random module
import random

# initialisation of random number generator
random.seed ()

#
# generating 10 random numbers of uniform distribution between 0 and 1
#

# generating 10 random numbers
print (f'10 random numbers of uniform distribution between 0 and 1')
for i in range (10):
    # generation of a random number of uniform distribution between 0 and 1
    r = random.random ()
    # printing generated random number
    print (f'  {r:15.13f}')
```

```
#
# generating 10 random numbers of uniform distribution between 100 and 200
#

# parameters
a = 100.0
b = 200.0

# generating 10 random numbers
print (f'10 random numbers of uniform distribution between {a} and {b}')
for i in range (10):
    # generation of a random number of uniform distribution between 100 and 200
    r = random.uniform (a, b)
    # printing generated random number
    print (f'  {r:15.11f}')

#
# generating 10 random numbers of Gaussian dist. of mean=100 and stddev=10
#

# parameters
mean   = 100.0
stddev = 10.0

# generating 10 random numbers
print (f'10 random numbers of Gaussian distribution', \
       f'of mean={mean} and stddev={stddev}')
for i in range (10):
    # generation of a random number of Gaussian distribution
    r = random.gauss (mean, stddev)
    # printing generated random number
    print (f'  {r:15.11f}')
```

```
% ./ai202209_s02_07.py
10 random numbers of uniform distribution between 0 and 1
  0.6301683894726
  0.4347830738973
  0.6234729750362
  0.4507141773723
  0.0559711586006
  0.1081476490463
  0.6800881379308
  0.5488567711388
  0.4144432760744
  0.9213050966332
10 random numbers of uniform distribution between 100.0 and 200.0
  187.52612233052
  105.28308484257
  188.06221779568
  138.55142129547
  145.23785051335
  124.83845056176
  171.15346854476
  186.27473074614
  128.05984806011
  199.58086360753
10 random numbers of Gaussian distribution of mean=100.0 and stddev=10.0
```

```
 99.40295348213
 96.02905973811
 85.37040369223
103.84093008636
 84.68432772055
106.87070997422
 97.15963763698
 91.11332510963
 97.32909104157
107.10278852741
```

Try following practice.

Practice 02-08

Make your own Python script to generate a set of random numbers of a distribution of your choice. Print generated random numbers. Mention which distribution you have chosen.

7 Using `statistics` module

The `statistics` module can be used to calculate statistical values, such as mean, median, mode, and standard deviation. Try following example.

Python Code 9: ai202209_s02_08.py

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

#
# Time-stamp: <2022/09/18 11:47:09 (CST) daisuke>
#

# importing statistics module
import statistics

# generation of a synthetic data set
dataset1 = [ 6.0, 7.0, 8.0, 9.0, 9.0, \
             10.0, 10.0, 11.0, 11.0, 11.0, \
             12.0, 13.0, 14.0]

# printing data set
print (f'dataset1:\n{dataset1}')

# calculation of mean
mean = statistics.fmean (dataset1)

# printing calculated mean
print (f'mean of dataset1   = {mean}')

# calculation of median
median = statistics.median (dataset1)

# printing calculated median
print (f'median of dataset1 = {median}')

# calculation of mode
mode = statistics.mode (dataset1)

# printing calculated mode
print (f'mode of dataset1   = {mode}')
```

```
# calculation of sample variance
var = statistics.variance (dataset1)

# printing calculated sample variance
print (f'sample variance of dataset1                = {var}')

# calculation of population variance
pvar = statistics.pvariance (dataset1)

# printing calculated population variance
print (f'population variance of dataset1           = {pvar}')

# calculation of sample standard deviation
stddev = statistics.stdev (dataset1)

# printing calculated sample variance
print (f'sample standard deviation of dataset1     = {stddev}')

# calculation of population standard deviation
pstddev = statistics.pstdev (dataset1)

# printing calculated population standard deviation
print (f'population standard deviation of dataset1 = {pstddev}')
```

```
% ./ai202209_s02_08.py
dataset1:
[6.0, 7.0, 8.0, 9.0, 9.0, 10.0, 10.0, 11.0, 11.0, 11.0, 12.0, 13.0, 14.0]
mean of dataset1   = 10.076923076923077
median of dataset1 = 10.0
mode of dataset1   = 11.0
sample variance of dataset1                = 5.243589743589744
population variance of dataset1            = 4.840236686390533
sample standard deviation of dataset1      = 2.2898885875932358
population standard deviation of dataset1 = 2.2000537917038603
```

Here are two more examples.

Python Code 10: ai202209_s02_09.py

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

#
# Time-stamp: <2022/09/18 11:49:37 (CST) daisuke>
#

# importing statistics module
import statistics

# importing random module
import random

# number of data in dataset2
ndata = 1000

# making an empty list for a synthetic dataset
dataset2 = []
```

```
# distribution parameters
a = 100.0
b = 200.0

# generation of a synthetic dataset using random number generator
for i in range (ndata):
    # generating a random number
    r = random.uniform (a, b)
    # appending generated random number to list
    dataset2.append (r)

# printing generated dataset
print (f'dataset2:')
for i in range (ndata):
    if ( (i > 4) and (i < ndata - 1) ):
        continue
    elif (i == ndata - 1):
        print (f'  ...')
    print (f'  {dataset2[i]}')

# number of data in dataset2
print (f'number of data in dataset2 = {len (dataset2)}')

# calculation of mean, median, and standard deviation
mean   = statistics.fmean (dataset2)
median = statistics.median (dataset2)
stddev = statistics.stdev (dataset2)

# printing mean, median, and standard deviation
print (f'mean of dataset2   = {mean}')
print (f'median of dataset2 = {median}')
print (f'stddev of dataset2 = {stddev}')
```

```
% ./ai202209_s02_09.py
dataset2:
  139.6299855136372
  104.08025822584986
  122.17068152495398
  153.11040268810083
  151.83789688509032
  ...
  134.56515974053795
number of data in dataset2 = 1000
mean of dataset2   = 151.41987013108977
median of dataset2 = 152.13600842866998
stddev of dataset2 = 29.232379344160876
```

Python Code 11: ai202209_s02_10.py

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

#
# Time-stamp: <2022/09/18 11:51:01 (CST) daisuke>
#

# importing statistics module
import statistics
```

```
# importing random module
import random

# number of data in dataset2
ndata = 1000

# making an empty list for a synthetic dataset
dataset3 = []

# distribution parameters
mean   = 100.0
stddev = 10.0

# generation of a synthetic dataset using random number generator
for i in range (ndata):
    # generating a random number
    r = random.gauss (mean, stddev)
    # appending generated random number to list
    dataset3.append (r)

# printing generated dataset
print (f'dataset3:')
for i in range (ndata):
    if ( (i > 4) and (i < ndata - 1) ):
        continue
    elif (i == ndata - 1):
        print (f'  ...')
    print (f'  {dataset3[i]}')

# number of data in dataset3
print (f'number of data in dataset3 = {len (dataset3)}')

# calculation of mean, median, and standard deviation
mean   = statistics.fmean (dataset3)
median = statistics.median (dataset3)
stddev = statistics.stdev (dataset3)

# printing mean, median, and standard deviation
print (f'mean of dataset3   = {mean}')
print (f'median of dataset3 = {median}')
print (f'stddev of dataset3 = {stddev}')
```

```
% ./ai202209_s02_10.py
dataset3:
  115.09241448186826
  88.13585177869867
  109.02792509926293
  109.00245865843561
  103.61230230972613
  ...
  91.88777069261927
number of data in dataset3 = 1000
mean of dataset3   = 100.54686367639589
median of dataset3 = 100.06318877161601
stddev of dataset3 = 10.082603921266333
```

Try following practice.

Practice 02-09

Make your own Python script to generate 10,000 random numbers of Gaussian distribution of mean 1000.0 and standard deviation 100.0. Calculate mean, median, and standard deviation. Print results of your calculation.

8 Using `decimal` module

Computers are using binary numbers for calculations, and calculations of floating point numbers are not exact in general. The `decimal` module provides correctly rounded floating point calculations.

First, try following example.

Python Code 12: ai202209_s02_11.py

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

#
# Time-stamp: <2022/09/18 14:41:42 (CST) daisuke>
#

# two floating point numbers "a" and "b"
a = 1.2
b = 2.4

# calculation of c = a + b
c = a + b

# printing result of calculation
print (f'{a} + {b}             = {c}')

# the other calculations
d = 1.1
e = 1.1
f = 1.1
g = d + e
h = d + e + f
i = d + e + f - 3.3
print (f'{d} + {e}             = {g}')
print (f'{d} + {e} + {f}       = {h}')
print (f'{d} + {e} + {f} - 3.3 = {i}')
```

```
% ./ai202209_s02_11.py
1.2 + 2.4             = 3.5999999999999996
1.1 + 1.1             = 2.2
1.1 + 1.1 + 1.1       = 3.3000000000000003
1.1 + 1.1 + 1.1 - 3.3 = 4.440892098500626e-16
```

As seen above, result of a calculation $1.2 + 2.4$ by Python is not 3.6, but it is 3.5999999999999996. To obtain correct value of 3.6, `decimal` module can be used. Try following example.

Python Code 13: ai202209_s02_12.py

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

#
# Time-stamp: <2022/09/18 15:01:14 (CST) daisuke>
#

# importing decimal module
```

```
import decimal

# two numbers "a" and "b" using decimal module
a = decimal.Decimal ('1.2')
b = decimal.Decimal ('2.4')

# calculation of c = a + b
c = a + b

# printing result of calculation
print (f'{a} + {b}             = {c}')

# the other calculations
d = decimal.Decimal ('1.1')
e = decimal.Decimal ('1.1')
f = decimal.Decimal ('1.1')
g = d + e
h = d + e + f
i = d + e + f - decimal.Decimal ('3.3')
print (f'{d} + {e}             = {g}')
print (f'{d} + {e} + {f}       = {h}')
print (f'{d} + {e} + {f} - 3.3 = {i}')
```

```
% ./ai202209_s02_12.py
./ai202209_s02_12.py
1.2 + 2.4             = 3.6
1.1 + 1.1             = 2.2
1.1 + 1.1 + 1.1       = 3.3
1.1 + 1.1 + 1.1 - 3.3 = 0.0
```

Next, try following.

Python Code 14: ai202209_s02_13.py

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

#
# Time-stamp: <2022/09/18 15:12:51 (CST) daisuke>
#

# importing decimal module
import decimal

# calculation of sqrt (2.0)
a = decimal.Decimal ('2.0')
b = a.sqrt ()

# result of calculation
print (f'a = {a}')
print (f'b = sqrt (a) = sqrt ({a}) = {b}')

# calculation of log10 (12.3)
c = decimal.Decimal ('12.3')
d = c.log10 ()

# result of calculation
print (f'c = {c}')
print (f'd = log10 (c) = log10 ({c}) = {d}')
```

```
% ./ai202209_s02_13.py
a = 2.0
b = sqrt (a) = sqrt (2.0) = 1.41421356237309504880168872 4
c = 12.3
d = log10 (c) = log10 (12.3) = 1.089905111439397931804439753
```

Calculate the value of π using Leibniz formula.

Python Code 15: ai202209_s02_14.py

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

#
# Time-stamp: <2022/09/18 15:49:50 (CST) daisuke>
#

# importing decimal module
import decimal

# importing math module
import math

# precision
decimal.getcontext ().prec = 50

# first 50 digits of pi
pi50 = decimal.Decimal ('3.14159265358979323846264338327950288419716939937 51')

# number of terms to calculate
n = 10**6

# initial value of pi
pi = decimal.Decimal ('0.0')

# calculation of pi using Leibniz formula
for i in range (n):
    pi += decimal.Decimal ('4.0') / (2 * i + 1) * (-1)**i

# printing math.pi, pi by Leibniz formula, first 50 digits of pi
print (f'math.pi                = {math.pi:51.49f}')
print (f'pi by Leibniz formula = {pi}')
print (f'first 50 digits of pi = {pi50}')
```

```
% ./ai202209_s02_14.py
math.pi                = 3.14159265358979311599796346854418516159057617187 50
pi by Leibniz formula = 3.14159165358979323871264338327919038419717035248 91
first 50 digits of pi = 3.14159265358979323846264338327950288419716939937 51
```

Increase the number of terms, and run the script. It may take 10 to 20 seconds for this calculation.

Python Code 16: ai202209_s02_15.py

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

#
# Time-stamp: <2022/09/18 15:50:14 (CST) daisuke>
#
```

```
# importing decimal module
import decimal

# importing math module
import math

# precision
decimal.getcontext().prec = 50

# first 50 digits of pi
pi50 = decimal.Decimal('3.14159265358979323846264338327950288419716939937510')

# number of terms to calculate
n = 10**7

# initial value of pi
pi = decimal.Decimal('0.0')

# calculation of pi using Leibniz formula
for i in range(n):
    pi += decimal.Decimal('4.0') / (2 * i + 1) * (-1)**i

# printing math.pi, pi by Leibniz formula, first 50 digits of pi
print(f'math.pi                = {math.pi:51.49f}')
print(f'pi by Leibniz formula = {pi}')
print(f'first 50 digits of pi = {pi50}')
```

```
% ./ai202209_s02_15.py
math.pi                = 3.14159265358979311599796346854418516159057617187500
pi by Leibniz formula = 3.14159255358979323846289338327950288107216939949520
first 50 digits of pi = 3.14159265358979323846264338327950288419716939937510
```

Try following practice.

Practice 02-10

Modify the script `ai202209_s02_15.py`, and calculate the value of π by using 10^8 terms of Leibniz formula. Show the result of your calculation.

9 Using `urllib` module

By using `urllib` module, files can be downloaded from remote sites. Try following example to download the first 1000 digits of π.

Python Code 17: ai202209_s02_16.py

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

#
# Time-stamp: <2022/09/18 16:43:46 (CST) daisuke>
#

# importing urllib module
import urllib.request

# importing ssl module
import ssl
```

```
# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

# URL of a resource
url_pi3 = 'https://newton.ex.ac.uk/research/qsystems/collabs/pi/pi3.txt'

# output file name
file_output = 'pi_1000.txt'

# printing status
print (f'Now, opening {url_pi3}...')

# opening URL
with urllib.request.urlopen (url_pi3) as fh_read:
    # reading data
    data_byte = fh_read.read ()

# printing status
print (f'Retrieved data from {url_pi3}!')

# printing type of "data_byte"
print (f'type of "data_byte" = {type (data_byte)}')

# converting raw byte data into string
data_str = data_byte.decode ('utf-8')

# printing type of "data_str"
print (f'type of "data_str"  = {type (data_str)}')

# printing status
print (f'Now, writing data to file "{file_output}"...')

# opening file for writing
with open (file_output, 'w') as fh_write:
    # writing data
    fh_write.write (data_str)

# printing status
print (f'Finished writing data to file "{file_output}"!')
```

```
% ./ai202209_s02_16.py
Now, opening https://newton.ex.ac.uk/research/qsystems/collabs/pi/pi3.txt...
Retrieved data from https://newton.ex.ac.uk/research/qsystems/collabs/pi/pi3.txt!
type of "data_byte" = <class 'bytes'>
type of "data_str"  = <class 'str'>
Now, writing data to file "pi_1000.txt"...
Finished writing data to file "pi_1000.txt"!
% ls -l pi_1000.txt
-rw-r--r--  1 daisuke  taiwan  1113 Sep 19 15:08 pi_1000.txt
```

Try following practice.

Practice 02-11

Make a Python script to download the Hipparcos catalogue. The Hipparcos catalogue is available at following URL.

- `https://cdsarc.cds.unistra.fr/ftp/I/239/hip_main.dat`

10 Using `pathlib` module

The `pathlib` module provides object-oriented interfaces for path and file handling. Try following example.

Python Code 18: ai202209_s02_17.py

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

#
# Time-stamp: <2022/09/19 07:46:22 (CST) daisuke>
#

# importing pathlib module
import pathlib

# file name
file_00 = 'ai202209_s02_00.py'

# making pathlib object
path_00 = pathlib.Path (file_00)

# existence check
exists_00 = path_00.exists ()
print (f'existence check:')
if (exists_00):
    print (f'  file "{file_00}" does exist.')
else:
    print (f'  file "{file_00}" does not exist.')

# printing parent, name, suffix, stem
print (f'parent, name, suffix, and stem:')
print (f'  parent of "{path_00}" = {path_00.parent}')
print (f'  name of "{path_00}"   = {path_00.name}')
print (f'  suffix of "{path_00}" = {path_00.suffix}')
print (f'  stem of "{path_00}"   = {path_00.stem}')

# finding directory contents of current working directory using .iterdir ()
path_cwd   = pathlib.Path ('.')
list_files = path_cwd.iterdir ()

# printing files in current working directory
print (f'files in current working directory:')
for file in sorted (list_files):
    print (f'  {file}')

# finding file information
file_pi = 'pi_1000.txt'
path_pi = pathlib.Path (file_pi)
info_pi = path_pi.stat ()

# printing information of file
print (f'file information of "pi_1000.txt":')
```

```
print (f'  file mode         : {oct (info_pi.st_mode)}')
print (f'  file size         : {info_pi.st_size} bytes')
print (f'  last modification : {info_pi.st_mtime} sec from 01/Jan/1970')

# opening file 'pi_1000.txt'
with path_pi.open () as fh:
    # reading file
    data = fh.read ()

# printing content of file 'pi_1000.txt'
print (f'---------- {file_pi} ----------')
print (data)
print (f'---------- {file_pi} ----------')
```

```
% ./ai202209_s02_17.py
existence check:
  file "ai202209_s02_00.py" does exist.
parent, name, suffix, and stem:
  parent of "ai202209_s02_00.py" = .
  name of "ai202209_s02_00.py"   = ai202209_s02_00.py
  suffix of "ai202209_s02_00.py" = .py
  stem of "ai202209_s02_00.py"   = ai202209_s02_00
files in current working directory:
  .ipynb_checkpoints
  ai202209_s02_00.py
  ai202209_s02_00.py~
  ai202209_s02_01.py
  ai202209_s02_01.py~
  ai202209_s02_02.py
  ai202209_s02_02.py~
  ai202209_s02_03.py
  ai202209_s02_03.py~
  ai202209_s02_04.py
  ai202209_s02_04.py~
  ai202209_s02_05.py
  ai202209_s02_05.py~
  ai202209_s02_06.py
  ai202209_s02_06.py~
  ai202209_s02_07.py
  ai202209_s02_07.py~
  ai202209_s02_08.py
  ai202209_s02_08.py~
  ai202209_s02_09.py
  ai202209_s02_09.py~
  ai202209_s02_10.py
  ai202209_s02_11.py
  ai202209_s02_11.py~
  ai202209_s02_12.py
  ai202209_s02_12.py~
  ai202209_s02_13.py
  ai202209_s02_13.py~
  ai202209_s02_14.py
  ai202209_s02_14.py~
  ai202209_s02_15.py
  ai202209_s02_16.py
  ai202209_s02_16.py~
  ai202209_s02_17.py
  ai202209_s02_17.py~
```

```
  ai202209_s02_18.py
  ai202209_s02_18.py~
  ai202209_s02_19.py
  ai202209_s02_19.py~
  ai202209_s02_20.py
  ai202209_s02_20.py~
  ai202209_s02_21.py
  ai202209_s02_21.py~
  ai202209_s02_22.py
  ai202209_s02_22.py~
  ai202209_s02_23.py
  ai202209_s02_23.py~
  ai202209_s02_24.py
  ai202209_s02_24.py~
  ai202209_s02_25.py
  ai202209_s02_25.py~
  ai202209_s02_26.py
  ai202209_s02_26.py~
  ai202209_s02_27.py
  ai202209_s02_27.py~
  ai202209_s02_28.py
  ai202209_s02_28.py~
  ai202209_s02_29.py
  ai202209_s02_29.py~
  ai202209_s02_30.py
  ai202209_s02_30.py~
  ai202209_s02_31.py
  ai202209_s02_31.py~
  ai202209_s02_32.py
  ai202209_s02_32.py~
  ai202209_s02_33.py
  ai202209_s02_33.py~
  ai202209_s02_34.py
  ai202209_s02_34.py~
  ai202209_s02_35.py
  ai202209_s02_35.py~
  bsc.data
  catalog
  catalog.gz
  pi_1000.txt
  pi_1000_2.txt
  test_argparse.ipynb
file information of "pi_1000.txt":
  file mode         : 0o100644
  file size         : 1113 bytes
  last modification : 1663571287.8680916 sec from 01/Jan/1970
---------- pi_1000.txt ----------
3.
1415926535 8979323846 2643383279 5028841971 6939937510
5820974944 5923078164 0628620899 8628034825 3421170679
8214808651 3282306647 0938446095 5058223172 5359408128
4811174502 8410270193 8521105559 6446229489 5493038196
4428810975 6659334461 2847564823 3786783165 2712019091
4564856692 3460348610 4543266482 1339360726 0249141273
7245870066 0631558817 4881520920 9628292540 9171536436
7892590360 0113305305 4882046652 1384146951 9415116094
3305727036 5759591953 0921861173 8193261179 3105118548
0744623799 6274956735 1885752724 8912279381 8301194912
9833673362 4406566430 8602139494 6395224737 1907021798
```

```
6094370277 0539217176 2931767523 8467481846 7669405132
0005681271 4526356082 7785771342 7577896091 7363717872
1468440901 2249534301 4654958537 1050792279 6892589235
4201995611 2129021960 8640344181 5981362977 4771309960
5187072113 4999999837 2978049951 0597317328 1609631859
5024459455 3469083026 4252230825 3344685035 2619311881
7101000313 7838752886 5875332083 8142061717 7669147303
5982534904 2875546873 1159562863 8823537875 9375195778
1857780532 1712268066 1300192787 6611195909 2164201989

---------- pi_1000.txt ----------
```

Try following example finding files and directories using `.glob ()` method.

Python Code 19: ai202209_s02_18.py

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

#
# Time-stamp: <2022/09/19 07:54:44 (CST) daisuke>
#

# importing pathlib module
import pathlib

# making pathlib object
path_cwd = pathlib.Path ('.')

# finding files recursively using .glob () method
list_files = path_cwd.glob ('**/*')

# printing files in current working directory and its sub-directories
print (f'files in current working directory and its sub-directories:')
for file in sorted (list_files):
    print (f'  {file}')
```

```
% ./ai202209_s02_18.py
files in current working directory and its sub-directories:
  .ipynb_checkpoints
  .ipynb_checkpoints/test_argparse-checkpoint.ipynb
  ai202209_s02_00.py
  ai202209_s02_00.py~
  ai202209_s02_01.py
  ai202209_s02_01.py~
  ai202209_s02_02.py
  ai202209_s02_02.py~
  ai202209_s02_03.py
  ai202209_s02_03.py~
  ai202209_s02_04.py
  ai202209_s02_04.py~
  ai202209_s02_05.py
  ai202209_s02_05.py~
  ai202209_s02_06.py
  ai202209_s02_06.py~
  ai202209_s02_07.py
  ai202209_s02_07.py~
  ai202209_s02_08.py
  ai202209_s02_08.py~
```

```
ai202209_s02_09.py
ai202209_s02_09.py~
ai202209_s02_10.py
ai202209_s02_11.py
ai202209_s02_11.py~
ai202209_s02_12.py
ai202209_s02_12.py~
ai202209_s02_13.py
ai202209_s02_13.py~
ai202209_s02_14.py
ai202209_s02_14.py~
ai202209_s02_15.py
ai202209_s02_16.py
ai202209_s02_16.py~
ai202209_s02_17.py
ai202209_s02_17.py~
ai202209_s02_18.py
ai202209_s02_18.py~
ai202209_s02_19.py
ai202209_s02_19.py~
ai202209_s02_20.py
ai202209_s02_20.py~
ai202209_s02_21.py
ai202209_s02_21.py~
ai202209_s02_22.py
ai202209_s02_22.py~
ai202209_s02_23.py
ai202209_s02_23.py~
ai202209_s02_24.py
ai202209_s02_24.py~
ai202209_s02_25.py
ai202209_s02_25.py~
ai202209_s02_26.py
ai202209_s02_26.py~
ai202209_s02_27.py
ai202209_s02_27.py~
ai202209_s02_28.py
ai202209_s02_28.py~
ai202209_s02_29.py
ai202209_s02_29.py~
ai202209_s02_30.py
ai202209_s02_30.py~
ai202209_s02_31.py
ai202209_s02_31.py~
ai202209_s02_32.py
ai202209_s02_32.py~
ai202209_s02_33.py
ai202209_s02_33.py~
ai202209_s02_34.py
ai202209_s02_34.py~
ai202209_s02_35.py
ai202209_s02_35.py~
bsc.data
catalog
catalog.gz
pi_1000.txt
pi_1000_2.txt
test_argparse.ipynb
```

Try following practice.

Practice 02-12

Make your own Python script to check the existence of a file.

11 Using `shutil` module

The `shutil` is a module for convenient file operations. Try following.

Python Code 20: ai202209_s02_19.py

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

#
# Time-stamp: <2022/09/19 08:06:52 (CST) daisuke>
#

# importing shutil module
import shutil

# importing pathlib module
import pathlib

# source file
file_source      = 'pi_1000.txt'

# destination file
file_destination = 'pi_1000_2.txt'

# making pathlib objects
path_source      = pathlib.Path (file_source)
path_destination = pathlib.Path (file_destination)

# existence check of file "pi_1000.txt"
if (path_source.exists ()):
    print (f'file "{file_source}" exists.')
else:
    print (f'file "{file_source}" DOES NOT exist.')

# existence check of file "pi_1000_2.txt"
if (path_destination.exists ()):
    print (f'file "{file_destination}" exists.')
else:
    print (f'file "{file_destination}" DOES NOT exist.')

# printing status
print (f'now, copying file from {file_source} to {file_destination}...')

# copying file
shutil.copy2 (file_source, file_destination)

# printing status
print (f'finished copying file from {file_source} to {file_destination}!')

# existence check of file "pi_1000.txt"
if (path_source.exists ()):
    print (f'file "{file_source}" exists.')
else:
```

```
    print (f'file "{file_source}" DOES NOT exist.')

# existence check of file "pi_1000_2.txt"
if (path_destination.exists ()):
    print (f'file "{file_destination}" exists.')
else:
    print (f'file "{file_destination}" DOES NOT exist.')

# printing status
print (f'now, removing file {file_destination}...')

# removing file "pi_1000_2.txt"
path_destination.unlink ()

# printing status
print (f'finished removing file {file_destination}!')

# existence check of file "pi_1000.txt"
if (path_source.exists ()):
    print (f'file "{file_source}" exists.')
else:
    print (f'file "{file_source}" DOES NOT exist.')

# existence check of file "pi_1000_2.txt"
if (path_destination.exists ()):
    print (f'file "{file_destination}" exists.')
else:
    print (f'file "{file_destination}" DOES NOT exist.')

# printing status
print (f'now, copying file from {file_source} to {file_destination}...')

# copying file again using pathlib objects
shutil.copy2 (path_source, path_destination)

# printing status
print (f'finished copying file from {file_source} to {file_destination}!')

# existence check of file "pi_1000.txt"
if (path_source.exists ()):
    print (f'file "{file_source}" exists.')
else:
    print (f'file "{file_source}" DOES NOT exist.')

# existence check of file "pi_1000_2.txt"
if (path_destination.exists ()):
    print (f'file "{file_destination}" exists.')
else:
    print (f'file "{file_destination}" DOES NOT exist.')
```

```
% ./ai202209_s02_19.py
file "pi_1000.txt" exists.
file "pi_1000_2.txt" DOES NOT exist.
now, copying file from pi_1000.txt to pi_1000_2.txt...
finished copying file from pi_1000.txt to pi_1000_2.txt!
file "pi_1000.txt" exists.
file "pi_1000_2.txt" exists.
now, removing file pi_1000_2.txt...
```

```
finished removing file pi_1000_2.txt!
file "pi_1000.txt" exists.
file "pi_1000_2.txt" DOES NOT exist.
now, copying file from pi_1000.txt to pi_1000_2.txt...
finished copying file from pi_1000.txt to pi_1000_2.txt!
file "pi_1000.txt" exists.
file "pi_1000_2.txt" exists.
% ls -l pi_1000*
-rw-r--r--  1 daisuke  taiwan  1113 Sep 19 15:08 pi_1000.txt
-rw-r--r--  1 daisuke  taiwan  1113 Sep 19 15:08 pi_1000_2.txt
% diff pi_1000*
```

Try following practice.

Practice 02-13

Make your own Python script to copy a file using `shutil` module.

Here is one more example for finding an executable.

Python Code 21: ai202209_s02_20.py

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

#
# Time-stamp: <2022/09/19 08:13:02 (CST) daisuke>
#

# importing shutil module
import shutil

# location of an executable named 'python'
location_python = shutil.which ('python')

# printing location of 'python'
print (f'location of "python"    = {location_python}')

# location of an executable named 'python2'
location_python2 = shutil.which ('python2')

# printing location of 'python2'
print (f'location of "python2"   = {location_python2}')

# location of an executable named 'python2.7'
location_python27 = shutil.which ('python2.7')

# printing location of 'python2.7'
print (f'location of "python2.7" = {location_python27}')

# location of an executable named 'python3'
location_python3 = shutil.which ('python3')

# printing location of 'python3'
print (f'location of "python3"   = {location_python3}')

# location of an executable named 'python3.9'
location_python39 = shutil.which ('python3.9')

# printing location of 'python3.9'
print (f'location of "python3.9" = {location_python39}')
```

```
# location of an executable named 'python4'
location_python4 = shutil.which ('python4')

# printing location of 'python4'
print (f'location of "python4"   = {location_python4}')
```

```
% ./ai202209_s02_20.py
location of "python"    = None
location of "python2"   = None
location of "python2.7" = /usr/pkg/bin/python2.7
location of "python3"   = /usr/local/bin/python3
location of "python3.9" = /usr/pkg/bin/python3.9
location of "python4"   = None
```

Try following practice.

Practice 02-14

Make your own Python script to find the location of the Python interpreter on your computer.

12 Using `subprocess` module

The shell commands can be executed by using `subprocess` module. Here is a simple example of using `subprocess` module.

Python Code 22: ai202209_s02_21.py

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

#
# Time-stamp: <2022/09/19 08:21:15 (CST) daisuke>
#

# importing subprocess module
import subprocess

# executing a command "cal"
subprocess.run ('cal')
```

```
% ./ai202209_s02_21.py
   September 2022
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

Try following practice.

Practice 02-15

Make your own Python script to execute `date` command.

Here is one more example to capture the output from the command.

Python Code 23: ai202209_s02_22.py

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

#
# Time-stamp: <2022/09/19 08:21:17 (CST) daisuke>
#

# importing subprocess module
import subprocess

# executing a command "cal 9 1752" and capturing output
result = subprocess.run ('cal 9 1752', shell=True, capture_output=True)

# stdout of command execution
output = result.stdout.decode ('utf-8')

# printing result of command execution
print (output)
```

```
% ./ai202209_s02_22.py
   September 1752
Su Mo Tu We Th Fr Sa
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

Try following practice.

Practice 02-16

Make your own Python script to execute `date` command, capture output from the command, and print it.

13 Using `datetime` module

In astronomy, we often need to deal with date/time. Here is an example of getting current date/time using `datetime` module.

Python Code 24: ai202209_s02_23.py

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

#
# Time-stamp: <2022/09/19 08:32:15 (CST) daisuke>
#

# importing datetime module
import datetime

# current time in local time
time_now_local = datetime.datetime.now ()

# printing result
print ("local time:       ", time_now_local)

# getting year, month, day, hour, minute, and second
YYYY = time_now_local.year
MM   = time_now_local.month
```

```
DD   = time_now_local.day
hh   = time_now_local.hour
mm   = time_now_local.minute
ss   = time_now_local.second + time_now_local.microsecond * 10**-6

# printing results
print (f'current date/time:',
       f'{YYYY:04d}/{MM:02d}/{DD:02d}T{hh:02d}:{mm:02d}:{ss:09.6f}')

# current time in UTC
time_now_utc   = datetime.datetime.now (tz=datetime.timezone.utc)

# printing result
print ("UTC:               ", time_now_utc)

# the other way to get current time in UTC
time_now_utc2  = datetime.datetime.utcnow ()

# printing result
print ("UTC:               ", time_now_utc2)
```

```
% ./ai202209_s02_23.py
local time:        2022-09-19 15:25:21.670577
current date/time: 2022/09/19T15:25:21.670577
UTC:               2022-09-19 07:25:21.670647+00:00
UTC:               2022-09-19 07:25:21.670696
```

Try following practice.

Practice 02-17

Make your own Python script to get the current local date/time in Hawaii.

14 Using `re` module

The regular expression is a powerful tool for manipulation of text data. In astronomy, we often need to read a catalogue file and find a set of data of our interest. The regular expression can be used for this purpose. The `re` module offers functionality of the regular expression and pattern matching.

14.1 Downloading Bright Star Catalogue

Use `urllib` module to download Yale Bright Star Catalogue.

Python Code 25: ai202209_s02_24.py

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

#
# Time-stamp: <2022/09/19 08:47:29 (CST) daisuke>
#

# importing urllib module
import urllib.request

# importing ssl module
import ssl

# allow insecure downloading
```

```
ssl._create_default_https_context = ssl._create_unverified_context

# URL of Yale Bright Star Catalogue
url_bsc = 'https://cdsarc.cds.unistra.fr/ftp/V/50/catalog.gz'

# output file name
file_bsc = 'catalog.gz'

# printing status
print (f'Now, opening {url_bsc}...')

# opening URL
with urllib.request.urlopen (url_bsc) as fh_read:
    # reading data
    data_byte = fh_read.read ()

# printing status
print (f'Retrieved data from {url_bsc}!')

# printing status
print (f'Now, writing data to file "{file_bsc}"...')

# opening file for writing
with open (file_bsc, 'wb') as fh_write:
    # writing data
    fh_write.write (data_byte)

# printing status
print (f'Finished writing data to file "{file_bsc}"!')
```

```
% ./ai202209_s02_24.py
Now, opening https://cdsarc.cds.unistra.fr/ftp/V/50/catalog.gz...
Retrieved data from https://cdsarc.cds.unistra.fr/ftp/V/50/catalog.gz!
Now, writing data to file "catalog.gz"...
Finished writing data to file "catalog.gz"!
% ls -l catalog.gz
-rw-r--r--  1 daisuke  taiwan  573921 Sep 19 15:30 catalog.gz
```

14.2 Checking existence of catalogue file

Use `pathlib` module to check the existence of the catalogue file.

Python Code 26: ai202209_s02_25.py

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

#
# Time-stamp: <2022/09/19 08:50:57 (CST) daisuke>
#

# importing pathlib module
import pathlib

# file of Yale Bright Star Catalogue
file_bsc = 'catalog.gz'

# making pathlib object
path_bsc = pathlib.Path (file_bsc)
```

```
# existence check of file
if (path_bsc.exists ()):
    print (f'File "{file_bsc}" exists.')
    print (f'Downloading of Yale Bright Star Catalogue was successfully done!')
else:
    print (f'File "{file_bsc}" DO NOT exist.')
    print (f'Download Yale Bright Star Catalogue!')
```

```
% ./ai202209_s02_25.py
File "catalog.gz" exists.
Downloading of Yale Bright Star Catalogue was successfully done!
```

14.3 Reading compressed file

Use `gzip` module to open and read the compressed catalogue file, and write decompressed data into a new file.

Python Code 27: ai202209_s02_26.py

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

#
# Time-stamp: <2022/09/19 09:03:46 (CST) daisuke>
#

# importing gzip module
import gzip

# file of Yale Bright Star Catalogue
file_bsc = 'catalog.gz'

# output file
file_output = 'bsc.data'

# opening a compressed file
with gzip.open (file_bsc, 'rb') as fh_read:
    # reading file
    data_byte = fh_read.read ()

# converting byte data into string
data_str = data_byte.decode ('utf-8')

# opening new file
with open (file_output, 'w') as fh_write:
    # writing catalogue into a new file
    fh_write.write (data_str)
```

```
% ./ai202209_s02_26.py
% ls -l bsc.data
-rw-r--r--  1 daisuke  taiwan  1704879 Sep 19 15:32 bsc.data
```

14.4 Pattern matching using regular expression

Use `re` module to try pattern matching. Find Vega (α Lyr) in Bright Star Catalogue.

Python Code 28: ai202209_s02_27.py

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

#
# Time-stamp: <2022/09/19 13:11:09 (CST) daisuke>
#

# importing re module
import re

# making a pattern for regular expression
pattern_vega = re.compile ('Alp Lyr')

# catalogue file of BSC
file_bsc = 'bsc.data'

# opening file for reading
with open (file_bsc, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # pattern matching using regular expression
        match_vega = re.search (pattern_vega, line)
        # if matched, print the line
        if (match_vega):
            # HR number
            HR    = line[0:4].strip ()
            # name
            name  = line[4:14].strip ()
            # RAh
            RAh   = line[75:77].strip ()
            # RAm
            RAm   = line[77:79].strip ()
            # RAs
            RAs   = line[79:83].strip ()
            # Decpm
            Decpm = line[83].strip ()
            # Decd
            Decd  = line[84:86].strip ()
            # Decm
            Decm  = line[86:88].strip ()
            # Decs
            Decs  = line[88:90].strip ()
            # Vmag
            Vmag  = line[102:107].strip ()
            # SpType
            SpType = line[127:147].strip ()
            # RA
            RA = f'{RAh}:{RAm}:{RAs}'
            # Dec
            Dec = f'{Decpm}{Decd}:{Decm}:{Decs}'
            # printing information about star
            print (f'{HR:4}  {name:10}  {RA}  {Dec}  {Vmag}  {SpType}')
```

```
% ./ai202209_s02_27.py
7001  3Alp Lyr    18:36:56.3  +38:47:01  0.03  A0Va
```

14.5 Finding O-type main-sequence stars

Use `re` module to find O-type main-sequence stars in Bright Star Catalogue.

Python Code 29: ai202209_s02_28.py

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

#
# Time-stamp: <2022/09/19 13:33:37 (CST) daisuke>
#

# importing re module
import re

# making a pattern for regular expression
pattern_type = re.compile ('O[.\d+]V')

# catalogue file of BSC
file_bsc = 'bsc.data'

# opening file for reading
with open (file_bsc, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # pattern matching using regular expression
        match_type = re.search (pattern_type, line)
        # if matched, print the information of the star
        if (match_type):
            # HR number
            HR    = line[0:4].strip ()
            # name
            name  = line[4:14].strip ()
            # RAh
            RAh   = line[75:77].strip ()
            # RAm
            RAm   = line[77:79].strip ()
            # RAs
            RAs   = line[79:83].strip ()
            # Decpm
            Decpm = line[83].strip ()
            # Decd
            Decd  = line[84:86].strip ()
            # Decm
            Decm  = line[86:88].strip ()
            # Decs
            Decs  = line[88:90].strip ()
            # Vmag
            Vmag  = line[102:107].strip ()
            # SpType
            SpType = line[127:147].strip ()
            # RA
            RA = f'{RAh}:{RAm}:{RAs}'
            # Dec
            Dec = f'{Decpm}{Decd}:{Decm}:{Decs}'
            # printing information about star
            print (f'{HR:4} {name:10} {RA:10} {Dec:9} {Vmag:5} {SpType}')
```

```
% ./ai202209_s02_28.py
2212 Del Pic    06:10:17.9 -54:58:07 4.81  B3III+09V
```

```
2456 15     Mon   06:40:58.7 +09:53:44 4.66  O7V((f))
2806              07:22:02.0 -08:58:45 6.43  O9V
6535              17:34:42.5 -32:34:54 5.70  O7V+O7V
6736 9     Sgr    18:03:52.4 -24:21:38 5.97  O4V((f))
7767              20:18:07.0 +40:43:56 5.84  O9V
8023              20:56:34.7 +44:55:30 5.96  O6V((f))
8406 14     Cep   22:02:04.6 +58:00:02 5.56  O9Vn
8622 10     Lac   22:39:15.7 +39:03:01 4.88  O9V
```

14.6 Finding M-type supergiants

Use `re` module to find M-type supergiants in Bright Star Catalogue.

Python Code 30: ai202209_s02_29.py

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

#
# Time-stamp: <2022/09/19 13:33:28 (CST) daisuke>
#

# importing re module
import re

# making a pattern for regular expression
pattern_type = re.compile ('M[.\d+][I][a-z*]')

# catalogue file of BSC
file_bsc = 'bsc.data'

# opening file for reading
with open (file_bsc, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # pattern matching using regular expression
        match_type = re.search (pattern_type, line)
        # if matched, print the information of the star
        if (match_type):
            # HR number
            HR    = line[0:4].strip ()
            # name
            name  = line[4:14].strip ()
            # RAh
            RAh   = line[75:77].strip ()
            # RAm
            RAm   = line[77:79].strip ()
            # RAs
            RAs   = line[79:83].strip ()
            # Decpm
            Decpm = line[83].strip ()
            # Decd
            Decd  = line[84:86].strip ()
            # Decm
            Decm  = line[86:88].strip ()
            # Decs
            Decs  = line[88:90].strip ()
            # Vmag
            Vmag  = line[102:107].strip ()
            # SpType
```

```
            SpType = line[127:147].strip ()
            # RA
            RA = f'{RAh}:{RAm}:{RAs}'
            # Dec
            Dec = f'{Decpm}{Decd}:{Decm}:{Decs}'
            # printing information about star
            print (f'{HR:4} {name:10} {RA:10} {Dec:9} {Vmag:5} {SpType}')
```

```
% ./ai202209_s02_29.py
1845  119     Tau 05:32:12.8 +18:35:40 4.38  M2Iab-Ib
2289  46Psi1Aur  06:24:53.9 +49:17:17 4.91  K5-M0Iab-Ib
2902             07:33:47.9 -14:31:26 4.97  M2Iabpe+B2V
3170             08:04:16.2 -32:40:30 5.31  M1Ib
3364             08:30:29.6 -36:43:16 6.69  M2Iab
6022             16:13:16.8 -53:40:18 5.83  M0Ib-II+F-G
6406  64Alp1Her  17:14:38.9 +14:23:25 3.48  M5Ib-II
6693             17:59:05.2 -30:15:11 5.16  M1Ib
8164             21:19:15.8 +58:37:25 5.66  M1Ibep+B2pe+B3V
8383             21:56:39.1 +63:37:32 4.91  M2Iaep+B8Ve
```

14.7 Finding very bright main-sequence stars

Use `re` module to find very bright main-sequence stars in Bright Star Catalogue.

Python Code 31: ai202209_s02_30.py

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

#
# Time-stamp: <2022/09/19 13:33:10 (CST) daisuke>
#

# importing re module
import re

# making a pattern for regular expression
pattern_type = re.compile ('[OBAFGKM][.\d+][V]')

# catalogue file of BSC
file_bsc = 'bsc.data'

# opening file for reading
with open (file_bsc, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # pattern matching using regular expression
        match_type = re.search (pattern_type, line)
        # if matched, print the information of the star
        if (match_type):
            # HR number
            HR    = line[0:4].strip ()
            # name
            name  = line[4:14].strip ()
            # RAh
            RAh   = line[75:77].strip ()
            # RAm
            RAm   = line[77:79].strip ()
            # RAs
```

```
            RAs   = line[79:83].strip ()
            # Decpm
            Decpm = line[83].strip ()
            # Decd
            Decd  = line[84:86].strip ()
            # Decm
            Decm  = line[86:88].strip ()
            # Decs
            Decs  = line[88:90].strip ()
            # Vmag
            Vmag  = line[102:107].strip ()
            # SpType
            SpType = line[127:147].strip ()
            # RA
            RA = f'{RAh}:{RAm}:{RAs}'
            # Dec
            Dec = f'{Decpm}{Decd}:{Decm}:{Decs}'

            # converting string into float
            try:
                Vmag_float = float (Vmag)
            except:
                Vmag_float = +999.999

            # if brighter than or equal to 1.0 mag, then print
            if (Vmag_float <= 1.0):
                # printing information about star
                print (f'{HR:4} {name:10} {RA:10} {Dec:9} {Vmag:5} {SpType}')
```

```
% ./ai202209_s02_30.py
472  Alp Eri    01:37:42.9 -57:14:12 0.46  B3Vpe
2491 9Alp CMa   06:45:08.9 -16:42:58 -1.46 A1Vm
5056 67Alp Vir  13:25:11.6 -11:09:41 0.98  B1III-IV+B2V
5459 Alp1Cen    14:39:35.9 -60:50:07 -0.01 G2V
6134 21Alp Sco  16:29:24.4 -26:25:55 0.96  M1.5Iab-Ib+B4Ve
7001 3Alp Lyr   18:36:56.3 +38:47:01 0.03  A0Va
7557 53Alp Aql  19:50:47.0 +08:52:06 0.77  A7V
```

Practice 02-18

Make your own Python script to find stars of your interest using `re` module.

15 Using `pint` module

The `pint` module allows us to do calculations of physical quantities in easy way. Here is an example of handling of angles.

Python Code 32: ai202209_s02_31.py

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

#
# Time-stamp: <2022/09/19 13:43:43 (CST) daisuke>
#

# importing pint module
import pint
```

```
# units
ur        = pint.UnitRegistry ()
u_deg     = ur.degree
u_rad     = ur.radian
u_arcmin = ur.arcmin
u_arcsec = ur.arcsec
u_mas     = ur.mas

# angle a in deg
a_deg = 180.0 * u_deg

# converting from deg to rad
a_rad = a_deg.to (u_rad)

# printing result
print (f'a = {a_deg}')
print (f'  = {a_rad}')

# angle b
b_deg     = 1.0 * u_deg
b_arcmin = b_deg.to (u_arcmin)
b_arcsec = b_deg.to (u_arcsec)
b_mas     = b_deg.to (u_mas)

# printing results
print (f'b = {b_deg}')
print (f'  = {b_arcmin}')
print (f'  = {b_arcsec}')
print (f'  = {b_mas}')
```

```
% ./ai202209_s02_31.py
a = 180.0 degree
  = 3.141592653589793 radian
b = 1.0 degree
  = 60.0 arcminute
  = 3599.9999999999995 arcsecond
  = 3600000.0 milliarcsecond
```

Here is one more example.

Python Code 33: ai202209_s02_32.py

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

#
# Time-stamp: <2022/09/19 13:47:02 (CST) daisuke>
#

# importing pint module
import pint

# units
ur        = pint.UnitRegistry ()
u_arcsec = ur.arcsec
u_mas     = ur.mas
u_pc      = ur.pc
u_au      = ur.au
```

```
u_m      = ur.metre

# parallax of Sirius
parallax = 374.5 * u_mas

# distance to Sirius
distance = 1.0 / parallax.to (u_arcsec) * u_pc * u_arcsec

# printing results
print (f'distance to Sirius = {distance}')
print (f'                   = {distance.to (u_au)}')
print (f'                   = {distance.to (u_m)}')
```

```
% ./ai202209_s02_32.py
distance to Sirius = 2.67022696929239 parsec
                   = 550773.8484525508 astronomical_unit
                   = 8.239459496574611e+16 meter
```

Here is an example of calculations of physical quantities.

Python Code 34: ai202209_s02_33.py

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

#
# Time-stamp: <2022/09/19 13:53:22 (CST) daisuke>
#

# importing pint module
import pint

# units
ur           = pint.UnitRegistry ()
u_km         = ur.km
u_sec        = ur.sec
u_km_per_sec = u_km / u_sec

# velocity
v = 300.0 * u_km_per_sec

# time
t = 10.0 * u_sec

# calculation of distance travelled
d = v * t

# printing result
print (f'velocity           = {v}')
print (f'time               = {t}')
print (f'distance travelled = {d}')
```

```
% ./ai202209_s02_33.py
velocity           = 300.0 kilometer / second
time               = 10.0 second
distance travelled = 3000.0 kilometer
```

Practice 02-19

Make your own Python script to the time needed for the light to travel from the Sun to the Earth. Use `pint` module.

16 Using `uncertainties` module

By using `uncertainties` module, the propagation of errors is conveniently calculated. Try following example.

Python Code 35: ai202209_s02_34.py

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

#
# Time-stamp: <2022/09/19 13:57:10 (CST) daisuke>
#

# importing uncertainties module
import uncertainties

# quantity "a": 6.0 +/- 0.3
a = uncertainties.ufloat (6.0, 0.3)

# quantity "b": 9.0 +/- 0.4
b = uncertainties.ufloat (9.0, 0.4)

# calculation of a + b
c = a + b

# printing value of "c"
print (f'c = a + b = {a} + {b} = {c}')
```

```
% ./ai202209_s02_34.py
c = a + b = 6.00+/-0.30 + 9.0+/-0.4 = 15.0+/-0.5
```

Here is one more example.

Python Code 36: ai202209_s02_35.py

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

#
# Time-stamp: <2022/09/19 14:04:54 (CST) daisuke>
#

# importing uncertainties module
import uncertainties

# quantity "a": 8.0 +/- 0.8
a = uncertainties.ufloat (8.0, 0.8)

# quantity "b": 4.0 +/- 0.6
b = uncertainties.ufloat (4.0, 0.6)

# calculation of a / b
c = a / b

# printing value of "c"
print (f'c = a / b = {a} / {b} = {c}')
```

```
% ./ai202209_s02_35.py
c = a / b = 8.0+/-0.8 / 4.0+/-0.6 = 2.0+/-0.4
```

Practice 02-20

Make your own Python script to calculate the propagation of errors. Use `uncertainties` module.

17 For your further reading

You can find detailed information about modules we have tried for this session on the official website of Python. Visit the official website of Python to learn more about useful modules of Python.

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

The official document of `pint` module can be found at following web page.

- `https://pint.readthedocs.io/en/stable/`

The official document of `uncertainties` module can be found at following web page.

- `https://pythonhosted.org/uncertainties/`

18 Assignment

1. Explain why you yield 3.3000000000000003 from the calculation of 1.1 + 2.2 using Python.
2. Use datetime module to check the current local time of following places. Show the source codes of your Python scripts.
   (a) Hawaii
   (b) Chile
   (c) Canary Island
3. The magnitude of star A is $m_A = 17.54 \pm 0.02$, and the magnitude of star B is $m_B = 18.93 \pm 0.03$. What is the magnitude difference of star A and star B and its uncertainty? Use `uncertainties` module to calculate them.
4. Hipparcos catalogue
   - Make a Python script to download Hipparcos catalogue from following web page.
   - Make a Python script to count the number of B-type main-sequence stars listed in Hipparcos catalogue. Use `re` module.
   - Make a Python script to count the number of G-type main-sequence stars listed in Hipparcos catalogue. Use `re` module.
   - Which is more numerous? B-type main-sequence stars? Or, G-type main-sequence stars? What is an implication from this?
   - Make a Python script to count the number of giants listed in Hipparcos catalogue. Use `re` module.
   - Make a Python script to count the number of main-sequence stars listed in Hipparcos catalogue. Use `re` module.
   - Which is more numerous? Giants? Or, main-sequence stars? What is an implication from this?
5. Calculation of π
   (a) Study about Gauss-Legendre formula. Describe the method of calculating π using Gauss-Legendre formula.
   (b) Make a Python script to calculate 1000 digits of π using Gauss-Legendre formula. Execute the script, and show the result. Show the source code of your Python script. Compare your result with the value shown on following web page.

- `http://mathshistory.st-andrews.ac.uk/HistTopics/1000_places.html`

(c) Calculate 10^6 digits of π using Gauss-Legendre formula. Make a Python script to compare your result with the value shown on following web page.

- `http://newton.ex.ac.uk/research/qsystems/collabs/pi/pi6.txt`

(d) (Optional) Calculate 10^9 digits of π. Make a Python script to compare your result with the value shown on following web page.

- `https://stuff.mit.edu/afs/sipb/contrib/pi/pi-billion.txt`