```
AlA euclid.pv
                                                                        A1E euclid.pv
                                                    # -----
# -----
"""EUCLID : compute the Euclidian division of two integer numbers"""
                                                    """EUCLID : compute the Euclidian division of two integer numbers"""
# -----
                                                    # -----
author = "Christophe Schlick"
                                                    author = "Christophe Schlick"
__version__ = "1.0" # without user input
                                                    version = "5.0" # split code into kernel and interface functions
                                                    __date__ = "2018-01-15"
date = "2018-01-15"
# -----
a. b = 34.12
                                                    def euclid(x,v):
print(a, '=', b, '*', a//b, '+', a%b)
                             # solution A
                                                     """return Euclidian decomposition: a = b*g + r"""
                                                     return f''(x) = \{y\} * \{x//y\} + \{x\%y\}''
print("%s = %s * %s + %s" % (a, b, a//b, a%b)) # solution B (much more flexible)
print(f"{a} = {b} * {a//b} + {a%b}")
                             # solution C (needs Python 3.6.x)
# -----
                                                    def parser(command):
                                                     """parse 'command' as two integers 'a,b' and return Euclidian decomposition"""
                    AlB euclid.pv
                                                     a, b = command.split(',')
# -----
                                                     a, b = int(a), int(b)
"""EUCLID : compute the Euclidian division of two integer numbers""
                                                     return euclid(a, b)
# -----
author = "Christophe Schlick"
                                                    def loop():
version = "2.0" # add interactive user input
                                                     """interaction loop for the "euclid" module"""
      = "2018-01-15"
 date
                                                     print("Note: enter empty line to stop interaction loop\n")
# -----
                                                     while True:
a = input("<> Value of numerator : ")
                                                      command = input("<> Enter numerator.denominator : ")
b = input("<> Value of denominator : ")
                                                      if command == '': break
print(type(a), type(b)) # 'a' and 'b' are strings
                                                      print(parser(command))
\#a, b = int(a), int(b)
                                                     print("See you later...")
print(type(a), type(b)) # 'a' and 'b' have been converted to integers
                                                    # ------
print(f"{a} = {b} * {a//b} + {a%b}")
                                                    if name == ' main ': # test whether this code is used as module or program
1000()
                                                    # -----
                    A1C euclid.pv
                                                                        A1F euclid.pv
# -----
"""EUCLID : compute the Euclidian division of two integer numbers"""
                                                    # -----
                                                    """EUCLID : compute the Euclidian division of two integer numbers"""
                                                    author = "Christophe Schlick"
version = "3.0" # add command line for user input
                                                    author = "Christophe Schlick"
      = "2018-01-15"
                                                    version = "6.0" # use 'userloop/convert/inspect' from the 'ezCLI' module
 date__
# -----
                                                    date__
                                                           = "2018-01-15"
command = input("<> Enter numerator,denominator : ")
a, b = command.split(',') # split command line at the ',' character
                                                    User input: <numerator>,<denominator> (where numerator:int, denominator:int > 0)
a, b = int(a), int(b)
                                                    App output: Euclidian division: numerator = denominator*q + r"""
print(f"{a} = {b} * {a//b} + {a%b}")
                                                    # ______
from ezCLI import *
                    A1D euclid.pv
                                                    def euclid(a,b):
# -----
                                                     """return Euclidian decomposition: a = b*g + r"""
"""EUCLID : compute the Euclidian division of two integer numbers"""
                                                     return f''(a) = \{b\} * \{a//b\} + \{a\%b\}''
# -----
 author = "Christophe Schlick"
                                                    def parser(command):
 version = "4.0" # add interaction loop
                                                     """parse 'command' as two integers 'a,b' and return Euclidian decomposition"""
      = "2018-01-15"
                                                     a, b = convert(command); #inspect()
# -----
                                                     return euclid(a, b)
                                                    # -----
print("Note: enter empty line to stop interaction loop\n")
                                                    if name == ' main ':
while True:
 command = input("<> Enter numerator.denominator : ")
                                                     userloop(parser, "Enter numerator, denominator") # user interaction loop
 if command == '': break # break loop when user enter an empty line
                                                    # -----
 a, b = command.split(',')
                                                                        A1G euclid.pv
 a, b = int(a), int(b)
 print(f"{a} = {b} * {a//b} + {a%b}")
                                                    # -----
print("See you later...")
                                                    """EUCLID : compute the Euclidian division of two integer numbers"""
# -----
                                                    # ------
                                                    _author__ = "Christophe Schlick"
```

```
version = "7.0" # add a few 'assert' statements to control user input
                                                            # -----
 date
        = "2018-01-15"
                                                             author = "Christophe Schlick"
 usage
                                                             version = "2.0" # use 'for' loop to generate string
                                                            date = "2018-01-15"
User input: <numerator>,<denominator> (where numerator:int, denominator:int > 0)
App output: Euclidian division: numerator = denominator*g + r"""
                                                            _usage__ = """
# -----
                                                            User input: <n> (where n:int > 0)
                                                            App output: sequence of square numbers from 1*1 to n*n"""
from ezCLI import *
                                                            # -----
def euclid(a.b):
                                                            from exCLT import *
 """return Euclidian decomposition: a = b*g + r"""
 return f''(a) = \{b\} * \{a//b\} + \{a\%b\}''
                                                            def square(n):
                                                             """return the square of 'n'"""
def parser(command):
                                                             return n*n
 """parse 'command' as two integers 'a,b' and return Euclidian decomposition"""
 command = convert(command); #inspect()
                                                            def squares(n):
 assert type(command) is tuple and len(command) == 2, "invalid syntax"
                                                             """return a string containing the 'n' first square numbers"""
                                                             lines = ''
 a, b = command; #inspect()
 assert type(a) is int, "numerator must be an integer"
                                                             for p in range(1, n+1):
 assert type(b) is int and b > 0, "denominator must be a positive integer"
                                                              lines += f''(p) * (p) = {square(p)}\n''; #inspect()
                                                             return lines.strip() # remove trailing newline character
 return euclid(a,b)
# -----
if name == ' main ':
                                                            def parser(command):
 userloop(parser, "Enter numerator, denominator") # user interaction loop
                                                             """parse 'command' as integer 'n' before calling 'squares(n)'"""
# -----
                                                             n = convert(command): #inspect()
                                                             assert type(n) is int and n > 0, "< n > must be a strictly positive integer"
                       A2A squares.py
                                                             return squares(n)
# -----
                                                             """SOUARES: print the sequence of square numbers from 1*1 to n*n"""
                                                            if name == ' main ':
# -----
                                                             userloop(parser, "Enter value for <n>")
author = "Christophe Schlick"
                                                            # -----
 version = "1.0" # use 'while' loop to generate string
                                                                                   A2C squares.py
date = "2018-01-15"
_usage__ = """
                                                            # -----
User input: <n> (where n:int > 0)
                                                            """SOUARES : print the sequence of square numbers from 1*1 to n*n"""
App output: sequence of square numbers from 1*1 to n*n""
                                                            author = "Christophe Schlick"
from ezCLT import *
                                                            version = "3.0" # use list comprehension then join into multi-line string
                                                             date__
                                                                    = "2018-01-15"
                                                                    = """
def square(n):
                                                            usage
 """return the square of 'n'"""
                                                            User input: <n> (where n:int > 0)
 return n*n
                                                            App output: sequence of square numbers from 1*1 to n*n""
                                                            from ezCLI import *
def squares(n):
 """return a string containing the 'n' first square numbers"""
                                                            # -----
 p, lines = 1, ''
                                                            def square(n):
 while (p \le n):
                                                             """return the square of 'n'"""
  lines += f''(p) * (p) = {square(p)} \n''
                                                             return n*n
                                                            # _____
  p += 1; #inspect()
                                                            def squares(n):
 return lines.strip() # remove trailing newline character
                                                             """return a string containing the 'n' first square numbers""
                                                             lines = [f"{p} * {p} = {square(p)}" for p in range(1,n+1)]; #inspect()
def parser(command):
 """parse 'command' as integer 'n' before calling 'squares(n)'"""
                                                             return '\n'.join(lines) # join all lines into a single multi-line string
 n = convert(command); #inspect()
 assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
                                                            def parser(command):
 return squares(n)
                                                             """parse 'command' as integer 'n' before calling 'squares(n)'"""
# ------
                                                             n = convert(command); #inspect()
if name == ' main ':
                                                             assert type(n) is int and n > 0, "< n > must be a strictly positive integer"
 userloop(parser, "Enter value for <n>")
                                                             return squares(n)
# -----
                                                             if name == ' main ':
                       A2B squares.pv
                                                             userloop(parser, "Enter value for <n>")
# -----
                                                            """SOUARES: print the sequence of square numbers from 1*1 to n*n"""
```

```
A3A multable.pv
                                                                                   A3C multable.pv
# -----
                                                            # -----
"""MILTABLE: print the multiplication table from 1*1 to n*n"""
                                                            """MILTABLE : print the multiplication table from 1*1 to n*n"""
# -----
                                                            # -----
author = "Christophe Schlick"
                                                            author = "Christophe Schlick"
__version__ = "1.0" # use two embedded 'for' loops to generate table
                                                            version = "3.0" # use 'grid' from the 'ezCLI' module
                                                            _date__ = "2018-01-15"
date = "2018-01-15"
_usage__ = """
                                                            usage = """
User input: <n> (where n:int > 0)
                                                            User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n""
                                                            App output: multiplication table from 1*1 to n*n""
                                                            from ezCLI import *
                                                            from ezCLI import *
# -----
                                                            # ______
                                                            def multable(n):
def multable(n):
 """return a string containing the multiplication table from 1*1 to n*n""
                                                              """return a string containing the multiplication table from 1*1 to n*n""
 lines = ''
                                                              # create 'table' as a matrix of integers
 for p in range(1, n+1):
                                                              table = [[p*q for q in range(1, n+1)] for p in range(1, n+1)]; #inspect()
                                                             return grid(table) # use the 'grid' function to format 'table' as a grid
  for g in range(1, n+1):
    lines += f"{p*q:3} "; #inspect() # string length is forced to 3 chars
  lines += '\n'
                                                            def parser(command):
 return lines.strip('\n') # remove trailing newline character
                                                              """parse 'command' as integer 'n' before calling 'multable(n)'"""
# ______
                                                             n = convert(command); #inspect()
def parser(command):
                                                             assert type(n) is int and n > 0. "<n> must be a strictly positive integer"
 """parse 'command' as integer 'n' before calling 'multable(n)'"""
                                                             return multable(n)
                                                            # ------
 n = convert(command); #inspect()
 assert type(n) is int and n > 0, "< n > must be a strictly positive integer"
                                                            if name == ' main ':
                                                             userloop(parser, "Enter value for <n>")
 return multable(n)
# -----
                                                            # -----
if name == ' main ':
                                                                                  A3D multable.pv
 userloop(parser, "Enter value for <n>")
# -----
                                                            """MULTABLE : print a (start, stop, step) slice of the multiplication table """
                      A3B multable.pv
                                                            # -----
# -----
                                                            author = "Christophe Schlick"
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
                                                            version = "4.0" # include (start, stop, step) parameters
                                                             date_
# -----
                                                                    = "2018-01-15"
 author = "Christophe Schlick"
                                                                     = """
                                                             usage__
 _version__ = "2.0" # use embedded list comprehension to generate table
                                                            User input: <start,stop,step>
 _date__ = "2018-01-15"
                                                                     - start:int = start value for the multiplication table
 usage = """
                                                                     - stop:int = stop value for the multiplication table
User input: <n> (where n:int > 0)
                                                                     - step:int = step value for the multiplication table
App output: multiplication table from 1*1 to n*n"""
                                                            App output: multiplication table from 'start*start' to 'stop*stop' """
# ------
                                                            # -----
from ezCLI import *
                                                            from ezCLI import *
# -----
                                                            # ______
def multable(n):
                                                            def multable(start, stop, step):
 """return a string containing the multiplication table from 1*1 to n*n""
                                                             """return a string containing a (start, stop, step) slice of multable"""
 # create 'table' as a matrix of 3-character strings
                                                              # create a list of integer values for the first row and first col
 table = [[f"{p*q:3}]" for q in range(1, n+1)] for p in range(1, n+1)]
                                                              values = [] if start == 1 else [1] # force the list to start with 1
 # join each line from 'table' into a single string
                                                             values += list(range(start, stop, step)); #inspect()
 lines = [' '.join(line) for line in table]; #inspect()
                                                              # create 'table' as a matrix of integers
 return '\n'.join(lines) # join all lines into a multi-line string
                                                             table = [[p*q for q in values] for p in values]; #inspect()
# -----
                                                              return grid(table) # use the 'grid' function to format 'table' as a grid
def parser(command):
 """parse 'command' as integer 'n' before calling 'multable(n)'"""
                                                            def parser(command):
 n = convert(command); #inspect()
                                                              """parse 'command' as (start, stop, step) values before calling 'multable'"""
 assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
                                                              start, stop, step = convert(command); #inspect()
 return multable(n)
                                                              assert type(start) is int and start > 0, f"{start!r} : invalid 'start' value"
# -----
                                                              assert type(stop) is int and stop >= start, f"{stop!r} : invalid 'stop' value"
if __name__ == '__main__':
                                                              assert type(step) is int and step > 0, f"{step!r} : invalid 'step' value"
                                                              return multable(start, stop, step)
 userloop(parser, "Enter value for <n>")
```

```
if name == ' main ':
 userloop(parser, "Enter <start, stop, step>")
# -----
                         A3E multable.pv
# -----
"""MULTABLE : print a (start, stop, step) slice of the multiplication table """
# -----
author = "Christophe Schlick"
 version = "5.0" # use 'parse' function with default parameter values
 date
         = "2018-01-15"
 usage
User input: ['start='<start>] ['stop='<stop>] ['step='step]
          - start:int = start value for multable (default = 1)
         - stop:int = stop value bound for multable (default = 10)
         - step:int = step value for multable loop (default = 1)
App output: multiplication table from 'start*start' to 'stop*stop' """
# -----
from ezCLI import *
def multable(start, stop, step):
 """return a string containing a (start.stop.step) slice of multable"""
 # create a list of integer values for the first row and first col
 values = [] if start == 1 else [1] # force the list to start with 1
 values += list(range(start, stop, step)); #inspect()
 # create 'table' as a matrix of integers
 table = [[p*q for q in values] for p in values]; #inspect()
 return grid(table) # use the 'grid' function to format 'table' as a grid
def parser(command):
 """parse 'command' as (start, stop, step) values before calling 'multable'"""
 default = 'start=1 stop=10 step=1' # default values for all arguments
 # parse 'command' and use default values for missing arguments
 args = parse(command, default): #inspect()
 # store all values from dictionary 'args' into variables
 start, stop, step = (args[n] for n in ('start', 'stop', 'step')); #inspect()
 assert type(start) is int and start > 0, f"{start!r} : invalid 'start' value"
 assert type(stop) is int and stop >= start, f"{stop!r} : invalid 'stop' value"
 assert type(step) is int and step > 0, f"{step!r} : invalid 'step' value"
 return multable(start, stop, step)
# -----
if name == ' main ':
 userloop(parser)
```

EXP-A - 4 - 2018/05/05 11:00