A1A_euclid.py

```
# ================================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0" # without user input
__date__    = "2018-01-15"
# ================================================================================
a, b = 34, 12
print(a, '=', b, '*', a//b, '+', a%b)          # solution A
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)
# ================================================================================
```

A1B_euclid.py

```
# ================================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "2.0" # add interactive user input
__date__    = "2018-01-15"
# ================================================================================
a = input("<> Value of numerator : ")
b = input("<> Value of denominator : ")
print(type(a), type(b)) # 'a' and 'b' are strings
#a, b = int(a), int(b)
print(type(a), type(b)) # 'a' and 'b' have been converted to integers
print(f"{a} = {b} * {a//b} + {a%b}")
# ================================================================================
```

A1C_euclid.py

```
# ================================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "3.0" # add command line for user input
__date__    = "2018-01-15"
# ================================================================================
command = input("<> Enter numerator,denominator : ")
a, b = command.split(',') # split command line at the ',' character
a, b = int(a), int(b)
print(f"{a} = {b} * {a//b} + {a%b}")
# ================================================================================
```

A1D_euclid.py

```
# ================================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "4.0" # add interaction loop
__date__    = "2018-01-15"
# ================================================================================
print("Note: enter empty line to stop interaction loop\n")
while True:
  command = input("<> Enter numerator,denominator : ")
  if command == '': break # break loop when user enter an empty line
  a, b = command.split(',')
  a, b = int(a), int(b)
  print(f"{a} = {b} * {a//b} + {a%b}")
print("See you later...")
# ================================================================================
```

A1E_euclid.py

```
# ================================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "5.0" # split code into kernel and interface functions
__date__    = "2018-01-15"
# ================================================================================
def euclid(x,y):
  """return Euclidian decomposition: a = b*q + r"""
  return f"{x} = {y} * {x//y} + {x%y}"
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' as two integers 'a,b' and return Euclidian decomposition"""
  a, b = command.split(',')
  a, b = int(a), int(b)
  return euclid(a, b)
# --------------------------------------------------------------------------------
def loop():
  """interaction loop for the "euclid" module"""
  print("Note: enter empty line to stop interaction loop\n")
  while True:
    command = input("<> Enter numerator,denominator : ")
    if command == '': break
    print(parser(command))
  print("See you later...")
# ================================================================================
if __name__ == '__main__': # test whether this code is used as module or program
  loop()
# ================================================================================
```

A1F_euclid.py

```
# ================================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "6.0" # use 'userloop/convert/inspect' from the 'ezCLI' module
__date__    = "2018-01-15"
__usage__   = """
User input: <numerator>,<denominator> (where numerator:int, denominator:int > 0)
App output: Euclidian division: numerator = denominator*q + r"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def euclid(a,b):
  """return Euclidian decomposition: a = b*q + r"""
  return f"{a} = {b} * {a//b} + {a%b}"
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' as two integers 'a,b' and return Euclidian decomposition"""
  a, b = convert(command); #inspect()
  return euclid(a, b)
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter numerator,denominator") # user interaction loop
# ================================================================================
```

A1G_euclid.py

```
# ================================================================================
"""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"
__usage__   = """
User input: <numerator>,<denominator> (where numerator:int, denominator:int > 0)
App output: Euclidian division: numerator = denominator*q + r"""
# ============================================================================
from ezCLI import *
# ----------------------------------------------------------------------------
def euclid(a,b):
  """return Euclidian decomposition: a = b*q + r"""
  return f"{a} = {b} * {a//b} + {a%b}"
# ----------------------------------------------------------------------------
def parser(command):
  """parse 'command' as two integers 'a,b' and return Euclidian decomposition"""
  command = convert(command); #inspect()
  assert type(command) is tuple and len(command) == 2, "invalid syntax"
  a, b = command; #inspect()
  assert type(a) is int, "numerator must be an integer"
  assert type(b) is int and b > 0, "denominator must be a positive integer"
  return euclid(a,b)
# ============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter numerator,denominator") # user interaction loop
# ============================================================================
```

A2A_squares.py

```
# ============================================================================
"""SQUARES : print the sequence of square numbers from 1*1 to n*n"""
# ============================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0" # use 'while' loop to generate string
__date__    = "2018-01-15"
__usage__   = """
User input: <n> (where n:int > 0)
App output: sequence of square numbers from 1*1 to n*n"""
# ============================================================================
from ezCLI import *
# ----------------------------------------------------------------------------
def square(n):
  """return the square of 'n'"""
  return n*n
# ----------------------------------------------------------------------------
def squares(n):
  """return a string containing the 'n' first square numbers"""
  p, lines = 1, ''
  while (p <= n):
    lines += f"{p} * {p} = {square(p)}\n"
    p += 1; #inspect()
  return lines.strip() # remove trailing newline character
# ----------------------------------------------------------------------------
def parser(command):
  """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"
  return squares(n)
# ============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter value for <n>")
# ============================================================================
```

A2B_squares.py

```
# ============================================================================
"""SQUARES : print the sequence of square numbers from 1*1 to n*n"""
# ============================================================================
__author__  = "Christophe Schlick"
__version__ = "2.0" # use 'for' loop to generate string
__date__    = "2018-01-15"
__usage__   = """
User input: <n> (where n:int > 0)
App output: sequence of square numbers from 1*1 to n*n"""
# ============================================================================
from ezCLI import *
# ----------------------------------------------------------------------------
def square(n):
  """return the square of 'n'"""
  return n*n
# ----------------------------------------------------------------------------
def squares(n):
  """return a string containing the 'n' first square numbers"""
  lines = ''
  for p in range(1, n+1):
    lines += f"{p} * {p} = {square(p)}\n"; #inspect()
  return lines.strip() # remove trailing newline character
# ----------------------------------------------------------------------------
def parser(command):
  """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"
  return squares(n)
# ============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter value for <n>")
# ============================================================================
```

A2C_squares.py

```
# ============================================================================
"""SQUARES : print the sequence of square numbers from 1*1 to n*n"""
# ============================================================================
__author__  = "Christophe Schlick"
__version__ = "3.0" # use list comprehension then join into multi-line string
__date__    = "2018-01-15"
__usage__   = """
User input: <n> (where n:int > 0)
App output: sequence of square numbers from 1*1 to n*n"""
# ============================================================================
from ezCLI import *
# ----------------------------------------------------------------------------
def square(n):
  """return the square of 'n'"""
  return n*n
# ----------------------------------------------------------------------------
def squares(n):
  """return a string containing the 'n' first square numbers"""
  lines = [f"{p} * {p} = {square(p)}" for p in range(1,n+1)]; #inspect()
  return '\n'.join(lines) # join all lines into a single multi-line string
# ----------------------------------------------------------------------------
def parser(command):
  """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"
  return squares(n)
# ============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter value for <n>")
# ============================================================================
```

A3A_multable.py

```
# ================================================================================
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0" # use two embedded 'for' loops to generate table
__date__    = "2018-01-15"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def multable(n):
  """return a string containing the multiplication table from 1*1 to n*n"""
  lines = ''
  for p in range(1, n+1):
    for q in range(1, n+1):
      lines += f"{p*q:3} "; #inspect() # string length is forced to 3 chars
    lines += '\n'
  return lines.strip('\n') # remove trailing newline character
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = convert(command); #inspect()
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return multable(n)
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A3B_multable.py

```
# ================================================================================
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "2.0" # use embedded list comprehension to generate table
__date__    = "2018-01-15"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def multable(n):
  """return a string containing the multiplication table from 1*1 to n*n"""
  # create 'table' as a matrix of 3-character strings
  table = [[f"{p*q:3}" for q in range(1, n+1)] for p in range(1, n+1)]
  # join each line from 'table' into a single string
  lines = [' '.join(line) for line in table]; #inspect()
  return '\n'.join(lines) # join all lines into a multi-line string
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = convert(command); #inspect()
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return multable(n)
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A3C_multable.py

```
# ================================================================================
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "3.0" # use 'grid' from the 'ezCLI' module
__date__    = "2018-01-15"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def multable(n):
  """return a string containing the multiplication table from 1*1 to n*n"""
  # create 'table' as a matrix of integers
  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
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = convert(command); #inspect()
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return multable(n)
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A3D_multable.py

```
# ================================================================================
"""MULTABLE : print a (start,stop,step) slice of the multiplication table"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "4.0" # include (start, stop, step) parameters
__date__    = "2018-01-15"
__usage__   = """
User input: <start,stop,step>
            - start:int = start value for the multiplication table
            - stop:int = stop value for the multiplication table
            - step:int = step value for the multiplication table
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'"""
  start, stop, step = convert(command); #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, "Enter <start,stop,step>")
# ============================================================================
```

A3E_multable.py

```
# ============================================================================
"""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)
# ============================================================================
```