A1A_euclid.py

```
# ==============================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # without user input
__date__    = "2022-11-12"
# ==============================================================================
a, b = 343, 125
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 (only Python >= 3.6)
# ==============================================================================
```

A1B_euclid.py

```
# ==============================================================================
"""EUCLID : compute the Euclidian division of two integer numbers"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # add interactive user input
__date__    = "2022-11-12"
# ==============================================================================
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 modified by Philippe Blasi"
__version__ = "3.0" # add command line for user input
__date__    = "2022-11-12"
# ==============================================================================
command = input("<> Enter <numerator> <denominator> : ")
a, b = command.split() # split command line into words
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 modified by Philippe Blasi"
__version__ = "4.0" # add interaction loop
__date__    = "2022-11-12"
# ==============================================================================
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 modified by Philippe Blasi"
__version__ = "5.0" # split code into kernel and interface functions
__date__    = "2022-11-12"
# ==============================================================================
def euclid(x:int,y:int) -> str:
  """return Euclidian decomposition: x = y*q + r"""
  return f"{x} = {y} * {x//y} + {x%y}"
# ------------------------------------------------------------------------------
def parser(command:str) -> str:
  """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() -> None:
  """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 modified by Philippe Blasi"
__version__ = "6.0" # use 'userloop/parse/inspect' from the 'ezCLI' module
__date__    = "2022-11-12"
__usage__   = """
User input: <numerator>,<denominator> (where numerator:int, denominator:int > 0)
App output: Euclidian division: numerator = denominator*q + r"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def euclid(x:int,y:int) -> str:
  """return Euclidian decomposition: x = y*q + r"""
  return f"{x} = {y} * {x//y} + {x%y}"
# ------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as two integers and return Euclidian decomposition"""
  a, b = parse(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 modified by Philippe Blasi"
```

```python
__version__ = "7.0" # add a few 'assert' statements to control user input
__date__    = "2022-11-12"
__usage__   = """
User input: <numerator>,<denominator> (where numerator:int, denominator:int > 0)
App output: Euclidian division: numerator = denominator*q + r"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def euclid(x:int,y:int) -> str:
  """return Euclidian decomposition: x = y*q + r"""
  return f"{x} = {y} * {x//y} + {x%y}"
# ------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as two integers and return Euclidian decomposition"""
  command = parse(command); #inspect()
  assert len(command) == 2, "two space-separated arguments are required"
  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

```python
# ==============================================================================
"""SQUARES : print the sequence of square numbers from 1*1 to n*n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # use 'while' loop to generate string
__date__    = "2022-11-12"
__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:int) -> int:
  """return the square of 'n'"""
  return n*n
# ------------------------------------------------------------------------------
def squares(n:int) -> str:
  """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:str) -> str:
  """parse 'command' as integer 'n' before calling 'squares(n)'"""
  n = parse(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

```python
# ==============================================================================
"""SQUARES : print the sequence of square numbers from 1*1 to n*n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # use 'for' loop to generate string
__date__    = "2022-11-12"
__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:int) -> int:
  """return the square of 'n'"""
  return n*n
# ------------------------------------------------------------------------------
def squares(n:int) -> str:
  """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:str) -> str:
  """parse 'command' as integer 'n' before calling 'squares(n)'"""
  n = parse(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

```python
# ==============================================================================
"""SQUARES : print the sequence of square numbers from 1*1 to n*n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "3.0" # use list comprehension then join into multi-line string
__date__    = "2022-11-12"
__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:int) -> int:
  """return the square of 'n'"""
  return n*n
# ------------------------------------------------------------------------------
def squares(n:int) -> str:
  """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:str) -> str:
  """parse 'command' as integer 'n' before calling 'squares(n)'"""
  n = parse(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>")
# ==============================================================================
```

A3Abis_multable.py

```
# ==============================================================================
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # use two embedded 'for' loops to generate table
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def multable(n:int) -> str:
  """return a string containing the multiplication table from 1*1 to n*n"""
  lines = ''
  size=len(str(n*n))+1
  for p in range(1, n+1):
    for q in range(1, n+1):
      lines += f"{(p*q):{size}d}"; #inspect() # string length is forced to 4 c……
……hars
    lines += '\n'
  return lines.strip('\n') # remove trailing newline character
# ------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = parse(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>")
# ==============================================================================
```

A3A_multable.py

```
# ==============================================================================
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # use two embedded 'for' loops to generate table
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def multable(n:int) -> str:
  """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):4d}"; #inspect() # string length is forced to 4 chars
    lines += '\n'
  return lines.strip('\n') # remove trailing newline character
# ------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = parse(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>")
# ==============================================================================
```

A3Bbis_multable.py

```
# ==============================================================================
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # use embedded list comprehension to generate table
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def multable(n:int) -> str:
  """return a string containing the multiplication table from 1*1 to n*n"""
  # create 'table' as a matrix of 3-character strings
  size=len(str(n*n))
  table = [[f"{(p*q):{size}d}" 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:str) -> str:
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = parse(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 modified by Philippe Blasi"
__version__ = "2.0" # use embedded list comprehension to generate table
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def multable(n:int) -> str:
  """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):3d}" 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:str) -> str:
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = parse(command); #inspect()
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return multable(n)
# ==============================================================================
```

```python
if __name__ == '__main__':
  userloop(parser, "Enter value for <n>")
# ==============================================================================
```

A3C_multable.py

```python
# ==============================================================================
"""MULTABLE : print the multiplication table from 1*1 to n*n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "3.0" # use 'grid' from the 'ezCLI' module
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int > 0)
App output: multiplication table from 1*1 to n*n"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def multable(n:int) -> str:
  """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:str) -> str:
  """parse 'command' as integer 'n' before calling 'multable(n)'"""
  n = parse(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

```python
# ==============================================================================
"""MULTABLE : print a (start,stop,step) slice of the multiplication table"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "4.0" # include (start, stop, step) parameters
__date__    = "2022-11-12"
__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:int, stop:int, step:int) -> str:
  """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 = list(range(start, stop+1, 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:str) -> str:
  """parse 'command' as (start,stop,step) values before calling 'multable'"""
  start, stop, step, *args = parse(command); #inspect()
  assert not args, "only three arguments allowed" # args must be empty
  assert type(start) is int and start > 0, "%s : invalid <start> value" % start
  assert type(stop) is int and stop >= start, "%s : invalid <stop> value" % stop
  assert type(step) is int and step > 0, "%s : invalid <step> value" % step
  return multable(start, stop, step)
# ==============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <start> <stop> <step>")
# ==============================================================================
```

A3D_multable_old.py

```python
# ==============================================================================
"""MULTABLE : print a (start,stop,step) slice of the multiplication table"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "4.0" # include (start, stop, step) parameters
__date__    = "2022-11-12"
__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:int, stop:int, step:int) -> str:
  """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:str) -> str:
  """parse 'command' as (start,stop,step) values before calling 'multable'"""
  start, stop, step, *args = parse(command); #inspect()
  assert not args, "only three arguments allowed" # args must be empty
  assert type(start) is int and start > 0, "%s : invalid <start> value" % start
  assert type(stop) is int and stop >= start, "%s : invalid <stop> value" % stop
  assert type(step) is int and step > 0, "%s : invalid <step> value" % step
  return multable(start, stop, step)
# ==============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <start> <stop> <step>")
# ==============================================================================
```