A1A_facto.py

```
# ================================================================================
"""FACTO : compute the sequence of factorial terms from 0! to n!"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # use iterative implementation for the factorial function
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int >= 0)
App output: sequence of factorial terms from 0! to n!"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def facto(n:int) -> int:
  """return n! (iterative implementation)"""
  value = 1
  for p in range(1, n+1):
    value *= p
  return value
# --------------------------------------------------------------------------------
def factos(n:int) -> str:
  """return a string containing the 'n' first factorial numbers"""
  lines = ''
  for p in range(0, n+1):
    lines += f"{p}! = {facto(p)}\n"
  return lines.strip() # remove trailing newline character
  # Alternative implementation using list comprehension
  # return '\n'.join(f"{p}! = {facto(p)}" for p in range(n+1))
# --------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as integer 'n' before calling 'factos(n)'"""
  n = parse(command)
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  return factos(n)
# ================================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A1B_facto.py

```
# ================================================================================
"""FACTO : compute the sequence of factorial terms from 0! to n!"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # use recursive implementation for the factorial function
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int >= 0)
App output: sequence of factorial values from 0! to n!"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def facto(n:int) -> int:
  """return n! (recursive implementation)"""
  if n == 0: return 1
  else: return n * facto(n-1)
  # Alternative implementation using conditional evaluation
  # return 1 if (n == 0) else n * facto(n-1)
# --------------------------------------------------------------------------------
def factos(n:int) -> str:
  """return a string containing the 'n' first factorial numbers"""
  lines = ''
  for p in range(0, n+1):
    lines += f"{p}! = {facto(p)}\n"
  return lines.strip() # remove trailing newline character
  # Alternative implementation using list comprehension
  # return '\n'.join(f"{p}! = {facto(p)}" for p in range(n+1))
# --------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as integer 'n' before calling 'factos(n)'"""
  n = parse(command)
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  return factos(n)
# ================================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A2A_fibo.py

```
# ================================================================================
"""FIBO : compute the n first terms of the Fibonacci sequence"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # use iterative implementation of the Fibonacci terms
__date__    = "2022-11-12"
__usage__   = """
User input: <n> (where n:int >= 0)
App output: sequence of the n first Fibonacci terms"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def fibo(n:int) -> int:
  """return the nth term of the Fibonacci sequence (iterative version)"""
  a, b = 0, 1
  for p in range(n):
    a, b = b, a+b
  return a
# --------------------------------------------------------------------------------
def fibos(n:int) -> str:
  """return a string containing the 'n' first terms of the Fibonacci sequence"""
  lines = ''
  for p in range(n+1):
    lines += f"fibo({p}) = {fibo(p)}\n"
  return lines.strip()
  # Alternative implementation using list comprehension
  #return '\n'.join(f"fibo({p}) = {fibo(p)}" for p in range(n+1))
# --------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as integer 'n' before calling 'fibos(n)'"""
  n = parse(command)
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  return fibos(n)
# ================================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A2B_fibo.py

```
# ================================================================================
"""FIBO : compute the n first terms of the Fibonacci sequence"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # use recursive implementation of the Fibonacci terms
__date__    = "2022-11-12"
__usage__   = """
```

```
User input: <n> (where n:int >= 0)
App output: sequence of the n first Fibonacci terms"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def fibo(n:int) -> int:
  """return the nth term of the Fibonacci sequence (recursive version)"""
  if n < 2: return n
  else: return fibo(n-2) + fibo(n-1)
  # Alternative implementation using conditional evaluation
  # return n if (n < 2) else fibo(n-2) + fibo(n-1)
# --------------------------------------------------------------------------------
def fibos(n:int) -> str:
  """return a string containing the 'n' first terms of the Fibonacci sequence"""
  lines = ''
  for p in range(n+1):
    lines += f"fibo({p}) = {fibo(p)}\n"
  return lines.strip()
  # Alternative implementation using list comprehension
  #return '\n'.join(f"fibo({p}) = {fibo(p)}" for p in range(n+1))
# --------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as integer 'n' before calling 'fibos(n)'"""
  n = parse(command)
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  return fibos(n)
# ================================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A3A_bino.py

```
# ================================================================================
"""BINO : compute the binomial coefficient C(n,p)"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # import factorial function from previous example
__date__    = "2022-11-12"
__usage__   = """
User input : <n>,<p> (where n:int >= 0, p:int >= 0)
App output : binomial coefficient C(n,p)"""
# ================================================================================
from ezCLI import *
from A1A_facto import facto
# --------------------------------------------------------------------------------
def bino(n:int, p:int) -> int:
  """compute the binomial coefficient C(n,p) using standard factorial"""
  return facto(n) // facto(p) // facto(n-p)
# --------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as 'n,p' before calling 'bino(n,p)'"""
  command = convert(command); #inspect()
  assert type(command) is tuple and len(command) == 2, "invalid syntax"
  n, p = command; #inspect()
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  assert type(p) is int and p >= 0, "<p> must be a positive integer"
  if p > n: n, p = p, n # swap 'n' and 'p'
  return f"bino({n},{p}) = {bino(n,p)}"
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <n>,<p>")
# ================================================================================
```

A3B_bino.py

```
# ================================================================================
"""BINO : compute the binomial coefficient C(n,p)"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # implement truncated factorial function for optimization
__date__    = "2022-11-12"
__usage__   = """
User input : <n>,<p> (where n:int >= 0, p:int >= 0)
App output : binomial coefficient C(n,p)"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def facto(n:int, m:int=0) -> int:
  """compute a truncated factorial term = m * (m+1) * (m+2) * ... * n"""
  value = 1
  for p in range(m+1, n+1):
    value *= p
  return value
# --------------------------------------------------------------------------------
def bino(n:int, p:int) -> int:
  """compute the binomial coefficient C(n,p) using truncated factorial"""
  m = max(p, n-p); return facto(n, m) // facto(n-m)
# --------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as 'n,p' before calling 'bino(n,p)'"""
  command = convert(command); #inspect()
  assert type(command) is tuple and len(command) == 2, "invalid syntax"
  n, p = command; #inspect()
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  assert type(p) is int and p >= 0, "<p> must be a positive integer"
  if p > n: n, p = p, n # swap 'n' and 'p'
  return f"bino({n},{p}) = {bino(n,p)}"
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <n>,<p>")
# ================================================================================
```

A4_framewords.py

```
# ================================================================================
"""FRAMEWORDS : frame each word of a user-provided list of words"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
__usage__   = """
User input : <word> [word ...] (where word:str without whitespaces)
App output : several framing modes for the list of words"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def framewords(command:str) -> str:
  """several framing modes for the list of words by using the 'grid' function"""
  one_cell = grid([[command]])
  one_row  = grid([command.split()])
  one_col  = grid([[word] for word in command.split()])
  one_grid = grid(command.split(), size=3)
  return '\n'.join((one_cell, one_row, one_col, one_grid))
# ================================================================================
if __name__ == '__main__':
  userloop(framewords, "Enter <word> [word ...]")
# ================================================================================
```

A5_binhex.py

```
# ================================================================================
"""BINHEX : show decimal, binary and hexadecimal representations for integers"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
__usage__   = """
User input : <value> [value...] where value:int is given in decimal
App output : show decimal, binary and hexa representation for all values"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def dec2bin(value:int) -> str:
  """return the binary representation for a (decimal) integer"""
  digits, binvalue = '01', [] if value else ['0']; #inspect()
  while value:
    binvalue.append(digits[value % 2])
    value //= 2; #inspect()
  binvalue.reverse(); #inspect()
  return '0B' + ''.join(binvalue)
# ,------------------------------------------------------------------------……
……---
def dec2hex(value:int) -> str:
  """return the hexadecimal representation for a (decimal) integer"""
  digits, hexvalue = '0123456789ABCDEF', [] if value else ['0']; #inspect()
  while value:
    hexvalue.append(digits[value % 16])
    value //= 16; #inspect()
  hexvalue.reverse(); #inspect()
  return '0X' + ''.join(hexvalue)
# --------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' into integers and apply binary/hexa conversion on them"""
  values = parse(command); #inspect()
  if type(values) is tuple:
    isinteger = [type(value) is int for value in values]; #inspect()
    assert all(isinteger), "all values must be integers"
    return '\n'.join(f"{n} = {dec2bin(n)} = {dec2hex(n)}" for n in values)
  else:
    assert type(values) is int, "all values must be integers"
    return f"{values} = {dec2bin(values)} = {dec2hex(values)}\n"
# ================================================================================
if __name__ == "__main__":
  userloop(parser, "Enter <value> [value...]")
# ================================================================================
```

A6A_muldiv.py

```
# ================================================================================
"""MULDIV : table of GCD/LCM (Greatest Common Divisor/Least Common Multiple)"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # use 'grid' function from ezCLI to format table
__date__    = "2022-11-12"
__usage__   = """
User input : <n> (where n:int > 0)
App output: table of GCD/LCM from 1 to n"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def gcd(p:int, q:int) -> int:
  """return Greatest Common Divisor between numbers 'p' and 'q'"""
  while q: p, q = q, p%q # compute GCD by using Euclids' algorithm
  return p
# --------------------------------------------------------------------------------
def lcm(p:int, q:int) -> int:
  """return Least Common Multiple between numbers 'p' and 'q'"""
  return p*q // gcd(p,q)
# --------------------------------------------------------------------------------
def cell(p:int, q:int) -> int:
  """return either the GCD or the LCM according to cell position (p,q)"""
  return lcm(p,q) if (p == 1) or (p > q) else gcd(p,q)
# --------------------------------------------------------------------------------
def muldiv(n:int) -> str:
  """return a grid containing the GCD/LCM table from 1 to n"""
  table = [[cell(p,q) for q in range(1, n+1)] for p in range(1, n+1)]
  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 'muldiv(n)'"""
  n = convert(command); #inspect()
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return muldiv(n)
# ================================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ================================================================================
```

A6B_muldiv.py

```
# ================================================================================
"""MULDIV : table of GCD/LCM (Greatest Common Divisor/Least Common Multiple)"""
# ================================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
__usage__   = """
User input : <n> (where n:int > 0)
App output: table of GCD/LCM from 1 to n"""
# ================================================================================
from ezCLI import *
from ezTK import *
# --------------------------------------------------------------------------------
def gcd(p:int, q:int) -> int:
  """return Greatest Common Divisor between numbers 'p' and 'q'"""
  while q: p, q = q, p%q # compute GCD by using Euclids' algorithm
  return p
# --------------------------------------------------------------------------------
def lcm(p:int, q:int) -> int:
  """return Least Common Multiple between numbers 'p' and 'q'"""
  return p*q // gcd(p,q)
# --------------------------------------------------------------------------------
def cell(p:int, q:int) -> int:
  """return either the GCD or the LCM according to cell position (p,q)"""
  return lcm(p,q) if (p == 1) or (p > q) else gcd(p,q)
# --------------------------------------------------------------------------------
def show(table:list) -> None:
  """show the content of 'table' in a tk window"""
  rows, cols = len(table), len(table[0]) # get size of table
  win = Win(title='MULDIV', font='Arial 14', fold=cols)
  for row in range(rows):
    for col in range(cols):
      Label(win, text=table[row][col], width=3, border=1)
  win.loop()
# --------------------------------------------------------------------------------
def muldiv(n:int) -> str:
```

```
  """return a grid containing the GCD/LCM table from 1 to n"""
  table = [[cell(p,q) for q in range(1, n+1)] for p in range(1, n+1)]
  show(table); return ''
# ------------------------------------------------------------------------------
def parser(command:str) -> str:
  """parse 'command' as integer 'n' before calling 'muldiv(n)'"""
  n = convert(command); #inspect()
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return muldiv(n)
# ==============================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ==============================================================================
```

A7A_pascal.py

```
# ==============================================================================
"""PASCAL : create Pascal's triangle (triangle of binomial terms)"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # use 'grid' function from ezCLI to format table
__date__    = "2022-11-12"
__usage__   = """
User input : <n> (where n:int > 0)
App output: Grid containing Pascal's triangle up to rank n"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def facto(n:int) -> int:
  """return n! (iterative implementation)"""
  value = 1
  for p in range(1, n+1):
    value *= p
  return value
# ------------------------------------------------------------------------------
def bino(n:int, p:int) -> int:
  """return the binomial coefficient C(n,p)"""
  return facto(n) // facto(p) // facto(n-p)
# ------------------------------------------------------------------------------
def pascal(n:int) -> str:
  """return a grid containing Pascal's triangle up to rank n"""
  table = [[bino(p,q) for q in range(0, p+1)] for p in range(0, n+1)]
  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 'pascal(n)'"""
  n = convert(command)
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return pascal(n)
# ==============================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ==============================================================================
```

A7B_pascal.py

```
# ==============================================================================
"""PASCAL : create Pascal's triangle (triangle of binomial terms)"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0"
__date__    = "2022-11-12"
__usage__   = """
User input : <n> (where n:int > 0)
App output: Grid containing Pascal's triangle up to rank n"""
# ==============================================================================
from ezCLI import *
from ezTK import *
# ------------------------------------------------------------------------------
def facto(n:int) -> int:
  """return n! (iterative implementation)"""
  value = 1
  for p in range(1, n+1):
    value *= p
  return value
# ------------------------------------------------------------------------------
def bino(n:int, p:int) -> int:
  """return the binomial coefficient C(n,p)"""
  return facto(n) // facto(p) // facto(n-p)
# ------------------------------------------------------------------------------
def show(table:list) -> None:
  """show the content of 'table' in a tk window"""
  rows, cols = len(table), len(table[-1]) # get size of table (use last row)
  width = len(str(max(table[-1]))) # find maximal width for table cells
  win = Win(title='MULDIV', font='Arial 14', fold=cols)
  for row in range(rows):
    for col in range(cols):
      text = table[row][col] if col <= row else '' # use '' for upper triangle
      Label(win, text=text, width=width, border=1)
  win.loop()
# ------------------------------------------------------------------------------
def pascal(n:int) -> str:
  """return a grid containing Pascal's triangle up to rank n"""
  table = [[bino(p,q) for q in range(0, p+1)] for p in range(0, n+1)]
  show(table); return ''
# ------------------------------------------------------------------------------
def parser(command:str) ->str:
  """parse 'command' as integer 'n' before calling 'pascal(n)'"""
  n = convert(command)
  assert type(n) is int and n > 0, "<n> must be a strictly positive integer"
  return pascal(n)
# ==============================================================================
if __name__ == "__main__":
  userloop(parser, "Enter value for <n>")
# ==============================================================================
```

A8A_prime.py

```
# ==============================================================================
"""PRIME : compute all prime numbers from 2 to n"""
# ==============================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "1.0" # test division with each previously found prime number
__date__    = "2022-11-12"
__usage__   = """
User input : <n> (where n:int > 1)
App output : sequence of prime numbers from 2 to n"""
# ==============================================================================
from ezCLI import convert, grid, write_csv
# ------------------------------------------------------------------------------
def wrap(items:list, n=15) -> list:
  """convert 1D list to 2D list by wrapping every 'n' items"""
  return [items[k:k+n] for k in range(0, len(items), n)]
# ------------------------------------------------------------------------------
def prime(n:int) -> list:
  """compute all prime numbers from 2 to n"""
  primes = [2]
  for p in range(3, n+1, 2): # loop over all odd numbers from 3 to 'n'
    for q in primes: # loop over all prime numbers already stored in 'primes'
```

```
      if p%q == 0: break # 'p' is not prime, so break loop and try next number
      if q*q > p: primes.append(p); break # 'p' is prime, so append to 'primes'
  return primes # return final list with all prime numbers up to 'n'
# ------------------------------------------------------------------------
def main():
  """manage user interaction loop"""
  print(f"{'-'*80}\n{__doc__}{__usage__}\n{'-'*80}") # show info (doc and usage)
  while True: # user interaction loop
    command = input("Enter value for <n> : ") # wait for user input
    if command == '': break # break interaction loop if user input is empty
    n = convert(command) # analyze user input and convert to appropriate type
    assert type(n) is int and n > 1, "<n> must be an integer and greater than 1"
    # --------------------------------------------------------------------
    primes = prime(n) # compute list of prime numbers up to 'n'
    print(grid(wrap(primes))) # show prime numbers in a grid with auto-wrapping
    # --------------------------------------------------------------------
    write_csv('primes.csv', primes) # write list on disk as CSV file
  print('See you later...')
# ========================================================================
if __name__ == "__main__":
  main()
# ========================================================================
```

A8B_prime.py

```
# ========================================================================
"""PRIME : compute all prime numbers from 2 to n"""
# ========================================================================
__author__  = "Christophe Schlick modified by Philippe Blasi"
__version__ = "2.0" # implement the Sieve of Erathostenes
__date__    = "2022-11-12"
__usage__   = """
User input : <n> (where n:int > 1)
App output : sequence of prime numbers from 2 to n"""
# ========================================================================
from ezCLI import convert, grid, write_csv
# ------------------------------------------------------------------------
def wrap(items:list, n=15) -> list:
  """convert 1D list to 2D list by wrapping every 'n' items"""
  return [items[k:k+n] for k in range(0, len(items), n)]
# ------------------------------------------------------------------------
def prime(n:int) -> list:
  """compute all prime numbers from 2 to n"""
  primes, notprimes = [2], set() # initialize 'list of primes, and set of
  for p in range(3, n+1, 2):
    if p not in notprimes: primes.append(p) # 'p' is prime so append to 'primes'
    for q in range(p*p, n+1, 2*p): #
      notprimes.add(q) # all odd multiples of 'p' are added to 'notprimes'
  return primes # return final list with all prime numbers up to 'n'
# ------------------------------------------------------------------------
def main():
  """command line interaction loop"""
  print(f"{'-'*80}\n{__doc__}{__usage__}\n{'-'*80}") # show info (doc and usage)
  while True: # user interaction loop
    command = input("Enter value for <n> : ") # wait for user input
    if command == '': break # break interaction loop if user input is empty
    n = convert(command) # analyze user input and convert to appropriate type
    assert type(n) is int and n > 1, "<n> must be an integer and greater than 1"
    # --------------------------------------------------------------------
    primes = prime(n) # compute list of prime numbers up to 'n'
    print(grid(wrap(primes))) # show prime numbers in a grid with auto-wrapping
    # --------------------------------------------------------------------
    write_csv('primes.csv', primes) # write list on disk as CSV file
  print('See you later...')
# ========================================================================
if __name__ == "__main__":
  main()
# ========================================================================
```