## B1_count.py

```
# ================================================================================
"""COUNT : print the number of lines, words and chars in a set of text files"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0"
__date__    = "2015-09-01"
__usage__   = """
User input : <filename> [filename ...]
App output : number of lines, words and chars for each provided file"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def count(name):
  """return the number of lines, words and chars stored in file 'name'"""
  text = read_txt(name) # read whole content of file 'name'
  line, word, char = len(text.split('\n')), len(text.split()), len(text)
  return f"{name} : lines = {line}, words = {word}, chars = {char}"
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' and return line/word/char counters for provided files"""
  names = parse(command); #inspect()
  return '\n'.join(count(name) for name in names)
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <filename> [filename ...]")
# ================================================================================
```

## B2_rilex.py

```
# ================================================================================
"""RILEX : print the lexical richness of a list of text files"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0"
__date__    = "2015-09-01"
__usage__   = """
User input : <filename> [filename ...]
App output : lexical richness of all provided files"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def rilex(name):
  """return the lexical richness of file 'name'"""
  words = read_txt(name).split() # split file content into words
  words = [word.upper() for word in words if len(word) > 3] # filter words
  histo = {} # initialize the word's histogram as an empty dictionary
  for word in words: histo[word] = histo.get(word,0) + 1
  return f"{name} : rilex = {len(histo)/max(1,len(words)):0.2}"
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' and return lexical richness of all provided files"""
  names = parse(command)
  return '\n'.join(rilex(name) for name in names)
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <filename> [filename ...]")
# ================================================================================
```

## B3_numlines.py

```
# ================================================================================
"""NUMLINES : print a list of text files with line numbering"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0"
__date__    = "2015-09-01"
__usage__   = """
User input : <filename> [filename ...]
App output : content of all provided files with line numbering"""
# ================================================================================
from ezCLI import *
# --------------------------------------------------------------------------------
def numlines(name):
  """add line numbering to all lines of file 'name'"""
  text = read_txt(name) # read whole content of file 'name'
  lines = text.split('\n')
  size = len(str(len(lines))) # max number of digits for counter
  lines = [f"{p+1:>{size}} - {line}" for p,line in enumerate(lines)]
  rule = '\n' + '\u2500'*80 + '\n' # horizontal rule (used as a separator)
  return name + ' : ' + rule + '\n'.join(lines) + rule
# --------------------------------------------------------------------------------
def parser(command):
  """parse 'command' and return content of provided files with line numbering"""
  names = parse(command); #inspect()
  return '\n'.join(numlines(name) for name in names)
# ================================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <filename> [filename ...]")
# ================================================================================
```

## B4_dice.py

```
# ================================================================================
"""DICE : display a graphical representation for a list of random dice rolls"""
# ================================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0"
__date__    = "2015-09-01"
__usage__   = """
User input : <number of dice> (must be an integer between 1 and 8)
App output : graphical representation for the provided number of dice rolls"""
# ================================================================================
from ezCLI import *
from random import randrange
# --------------------------------------------------------------------------------
def roll(n):
  """return a list of 'n' random dice rolls"""
  return [1 + randrange(6) for loop in range(n)]
# --------------------------------------------------------------------------------
def dice(rolls):
  """return a graphical representation for a list of random dice rolls"""
  # first create the dice font as a single string with 5 lines of 60 characters
  font = """
••••••••:••••••••:••••••••:••••••••:••••••••:••••••••
•       •:•     • •:•     • •:• •   • •:• •   • •:• •   • •
•   •   •:•       •:•   •   •:•       •:•   •   •:• •   • •
•       •:• •     •:• •     •:• •   • •:• •   • •:• •   • •
••••••••:••••••••:••••••••:••••••••:••••••••:••••••••
"""
  # then convert font into a 5x6 matrix by splitting string at '\n' and at ':'
  font = [line.split(':') for line in font.strip().split('\n')]
  # for each line, extract the chars of the current digit from the font matrix
  lines = [[line[roll-1] for roll in rolls] for line in font]
  # and finally, join everything as a multi-line string
  return '\n'.join(' '.join(line) for line in lines)
# --------------------------------------------------------------------------------
def parser(command):
```

```
  """parse 'command' and return the graphical representation for dice rolls"""
  n = convert(command)
  assert type(n) is int and 0<n<9, "number must be an integer between 1 and 8"
  return dice(roll(n)) # roll 'n' dice and return its graphical representation
# ============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter number of dice")
# ============================================================================
```

## B5A_fiboplus.py

```
# ============================================================================
"""FIBOPLUS : generate the Fibonacci sequence with several options"""
# ============================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0" # use 'parse' function with default values
__date__    = "2015-09-01"
__usage__   = """
User input : ['n =' <n>] ['u =' <u>] ['v =' <v>] ['file =' <filename>]
             - n:int = number of terms for the sequence (defaut = 10)
             - u:int = value of the first term (default = 0)
             - v:int = value of the second term (default = 1)
             - filename:str = name of output file (default = output on screen)
App ouput : Fibonacci sequence defined by user arguments

Note: to use all default values, simply enter <SPACE> as command"""
# ============================================================================
from ezCLI import *
# ----------------------------------------------------------------------------
def fibo(n, u, v):
  """compute the nth term of the Fibonacci sequence starting from (u,v)"""
  a, b = u, v
  for p in range(n):
    a, b = b, a+b
  return a
# ----------------------------------------------------------------------------
def fibos(n, u, v):
  """return the 'n' first terms of the Fibonacci starting from (u,v)"""
  return '\n'.join([f"fibo({p}) = {fibo(p,u,v)}" for p in range(n+1)])
# ----------------------------------------------------------------------------
def parser(command):
  """parse 'command' and return Fibonacci sequence of provided arguments"""
  default = 'n=10 u=0 v=1 file=' # 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
  n, u, v, file = (args[name] for name in ('n','u','v','file')); #inspect()
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  assert type(u) is int, "<u> must be an integer"
  assert type(v) is int, "<v> must be an integer"
  assert type(file) is str, "<filename> must be a string"
  if not file: return fibos(n, u, v) # show result on screen
  write_txt(file, fibos(n, u, v)) # write result in 'file'
  return f"Fibonacci sequence ({n} values) written in file '{file}'"
# ============================================================================
if __name__ == '__main__':
  userloop(parser)
# ============================================================================
```

## B5B_fiboplus.py

```
# ============================================================================
"""FIBOPLUS : generate the Fibonacci sequence with several options"""
# ============================================================================
__author__  = "Christophe Schlick"
__version__ = "2.0" # convert function 'fibo' into generator
__date__    = "2015-09-01"
__usage__   = """
User input : ['n =' <n>] ['u =' <u>] ['v =' <v>] ['file =' <filename>]
             - n:int = number of terms for the sequence (defaut = 10)
             - u:int = value of the first term (default = 0)
             - v:int = value of the second term (default = 1)
             - file:str = filename for output (default = output on screen)
App ouput : Fibonacci sequence defined by user arguments

Note: to use all default values, simply enter <SPACE> as command"""
# ============================================================================
from ezCLI import *
# ----------------------------------------------------------------------------
def fibo(n, u, v):
  """compute the nth term of the Fibonacci sequence starting from (u,v)"""
  a, b = u, v; yield a
  for p in range(n):
    a, b = b, a+b
    yield a
# ----------------------------------------------------------------------------
def fibos(n, u, v):
  """return the 'n' first terms of the Fibonacci starting from (u,v)"""
  return '\n'.join([f"fibo({p}) = {f}" for p,f in enumerate(fibo(n,u,v))])
# ----------------------------------------------------------------------------
def parser(command):
  """parse 'command' and return Fibonacci sequence of provided arguments"""
  default = "n=10 u=0 v=1 file=" # 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
  n, u, v, file = (args[name] for name in ('n','u','v','file')); #inspect()
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  assert type(u) is int, "<u> must be an integer"
  assert type(v) is int, "<v> must be an integer"
  assert type(file) is str, "<filename> must be a string"
  if not file: return fibos(n, u, v)
  write_txt(file, fibos(n, u, v))
  return "Fibonacci sequence (%s values) written in file %r" % (n, file)
# ============================================================================
if __name__ == '__main__':
  userloop(parser)
# ============================================================================
```

## B5C_fiboplus.py

```
# ============================================================================
"""FIBOPLUS : generate the Fibonacci sequence with several options"""
# ============================================================================
__author__  = "Christophe Schlick"
__version__ = "3.0" # compare speed of function and generator versions
__date__    = "2015-09-01"
__usage__   = """
User input : ['n =' <n>] ['u =' <u>] ['v =' <v>] ['file =' <filename>]
             - n:int = number of terms for the sequence (defaut = 10)
             - u:int = value of the first term (default = 0)
             - v:int = value of the second term (default = 1)
             - file:str = filename for output (default = output on screen)
App ouput : Fibonacci sequence defined by user arguments

Note: to use all default values, simply enter <SPACE> as command"""
# ============================================================================
from ezCLI import *
from B5B_fiboplus import fibos as gen_fibo # import generator implementation
```

```
from B5A_fiboplus import fibos as fct_fibo # import function implementation
# ------------------------------------------------------------------------------
def parser(command):
  """parse 'command' and return Fibonacci sequence of provided arguments"""
  default = "n=10 u=0 v=1 file=" # 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
  n, u, v, file = (args[name] for name in ('n','u','v','file')); #inspect()
  assert type(n) is int and n >= 0, "<n> must be a positive integer"
  assert type(u) is int, "<u> must be an integer"
  assert type(v) is int, "<v> must be an integer"
  assert type(file) is str, "<filename> must be a string"
  # 'timer' function measures timing for 1000 calls of the provided function
  timer('fct_fibo(n,u,v)'); timer('gen_fibo(n,u,v)'); return ''
# ==============================================================================
if __name__ == '__main__':
  userloop(parser)
# ==============================================================================
```

## B6_scramble.py

```
# ==============================================================================
"""SCRAMBLE : scramble the lines of a text file according to a password"""
# ==============================================================================
__author__  = "Christophe Schlick"
__version__ = "1.0"
__date__    = "2015-09-01"
__usage__   = """
User input : <password> <filename> [filename...]
App output : scramble the content of all 'filenames' according to 'password'
Note: use quotes if 'password' contains space characters"""
# ==============================================================================
from ezCLI import *
# ------------------------------------------------------------------------------
def scramble(password, name):
  """scramble the content of file 'name' by using characters of 'password'"""
  text, n = '', len(password) # the length of 'password' is needed for cycling
  for p, char in enumerate(read_txt(name)): # enumerate all chars from file
    # cycle around all characters in password, keep its 6 least significant bits
    # and use them to scramble each file character by applying an XOR operator
    code_char, code_pass = ord(char), 63 & ord(password[p % n])
    text += chr(code_char ^ code_pass) # scramble code with XOR operator
  write_txt(name, text) # replace file content by scrambled text
  return f"File {name} has been scrambled"
# ------------------------------------------------------------------------------
def parser(command):
  """extract arguments from 'command' before calling 'scramble()'"""
  password, *names = parse(command); #inspect()
  return '\n'.join(scramble(password, name) for name in names)
# ==============================================================================
if __name__ == '__main__':
  userloop(parser, "Enter <password> <filename> [filename...]")
# ==============================================================================
```