Author: Rusu Cosmin

Problem statement

Optional homework – implementation operations + conversions

- share in the final grade: 10%
- deadline for homework submission: the second week of December 2015

The application must implement algorithms for:

- arithmetic operations: addition, subtraction, multiplication and division by one digit, in a base p from the set {2,3,...,9,10,16}
- conversions of natural numbers between two bases p,q from {2,3,...,9,10,16} using the substitution method or successive divisions and rapid conversions between two bases p,q{2, 4, 8, 16}.

and must have a menu such that all operations and conversion methods to be verified separately.

The mark is computed as follows:

10%: by default

70% : the application (the author name will be found in code and will be visible at run too)

1p - algorithm for the method of successive divisions

1p - algorithm for the substitution method

1p – algorithm for conversion using 10 as an intermediate base

2p - rapid conversions (executable form) between two bases p,q from {2, 4, 8, 16}.

1p addition of two numbers in a base

1p subtraction of two numbers in a base

1p multiplication of a number by a digit in a base

1p division of a number by a digit in a base

1p code quality (indentation, use of comments, suggestive variables names)

20%: documentation

1p problem statement

1p sub-algorithm's diagram

1p used data type specification

3p specification and pseudo-code for the important algorithms used (input, output, preconditions, post-conditions -1p; pseudo-code 2p)

3p at least a set of test data for the complete application, more data sets where is needed

1p documentation clearness (structured, well written, ...)

December 2015

Author: Rusu Cosmin

Base Calculator - Feature list

F1	**Add** two numbers
F2	**Subtract** two numbers
F3	**Multiply** two numbers
F4	**Divide** a number by a digit in a base
F5	**Algorithm** for the method of successive divisions
F6	**Algorithm** for the substitution method
F7	**Algorithm** for conversion using 10 as an intermediate base
F8	**Rapid conversions** between two bases p,q from {2,4, 8, 16}.
F9	**User interface**
F10	**Tests**

Author: Rusu Cosmin

Work items/tasks

T1	Implement Integer class: the class to represent a 'number'
T2	Implement the operations on two Integer numbers: addition, subtraction, multiplication, division with one digit
T3	Implement the algorithms for conversion between two bases
T4	Implement user interface

Author: Rusu Cosmin

Project structure

The project is split into four parts(modules):

1. The application coordinator: app_coordinator.py
    - it starts the application and it let it run
    - it instantiates two objects:
        - a Tester instance to run the unit test
        - a Calculator instance and it runs the run() functions
2. UI module
    - Calculator class (calculator.py)
        - Class to control the whole UI interaction
        - Calculator.Menu - a static list containig the allowed operations a user can do
3. Tester module
    - Tester class (tester.py)
        - Class to test the unit tests for every operation and conversions
        - Tester.AllowedBases – a static list containing the allowed bases: [2, 3, 4, 5, 6, 7, 8, 9, 10, 16]
4. Model module
    - Integer class (integer.py)
        - Class to represent and encapsulate all the information for uniquely identifying a number in an arbitrary base
        - Properties:
            1. _base – the base of the number
            2. _length – the number of digits in the representation of our number in base _base
            3. _digits – the list of the number's digits. The less significant digits are on the lower index of the list
    - Exception class (exceptions.py)
        - Class to represent and encapsulate an Integer Exception

The code has specifications and every function describes what it does. For that, see the docs folder.

December 2015

Author: Rusu Cosmin

Work item T1

Implement and design the Integer (Number) class.

Chosen design:

```
class Integer:
    """
    Class to represent a number

    1. **self._base**       -the base of the number
    2. **self._length**     -the number of digits
    3. **self._digits**     -the list of the number's digits
                        -the less significant digits are on the lower index of the array
    """
```

Constructor:

```
def __init__(self, base, number_repr):
    '''
    Constructor for the Integer class
    :param base: an integer - the base in which the number is represented
    :param number_repr: a string - the representation of the number in that base
    :raises IntegerException if the base is not one of the allowed base
    :raises IntegerException if the number has less than one digit
    '''
```

To create a new Integer we use the constructor as follows:

var = Integer(base, representation)

Examples:

1. x = Integer(10, “123456”)
2. myNumber = Integer(16, “ABCDE1”)

Author: Rusu Cosmin

Work Item T2

Implement the operations on two Integer numbers: addition, subtraction, multiplication, division with one digit

Addition

```
def __add__(self, other):
    '''
    Function to implement the addition on two Integer objects
    :param other - the second operand of addition (the first one is self)
    :return an Integer representing the sum of self and other
    :raises ValueError if other is not of type Integer or if they are not in the same base
    '''
    if not isinstance(other, Integer):
        raise ValueError("Error - Addition on two different objects.")
    if self._base != other._base:
        raise ValueError("Error - Addition on two Integers with different bases.")
    base = self.getBase()
    new = Integer(base, repr(self))
    if len(new) < len(other):
        for i in range(0, len(other) - len(new)):
            new.append(0)
    else:
        for i in range(0, len(new)- len(other)):
            other.append(0)
    t = 0 #will handle the transport
    for i in range(0, new._length):
        val = new[i] + other[i] + t
        new[i] = val % base
        t = val // base
    if t:
        new.append(t)
    return new
```

Our algorithm is quite straightforward:

- We use the advantage of how we stored the digits of a number. In the first stage, we append insignificant zeroes to the smallest number. If both of the numbers have the same number of digits, than we do not need to append any leading zeros.
- Then we simply do the addition like we would do it ‘on the paper’
- Finally, we have to add the last transport. Ex: 999 + 999 = 1998

Subtraction

```
def __sub__(self, other):
    '''
    Function to implement the subtraction of an Integer from self.
    :param other - the subtrahend of subtraction (the Minuend is self)
    :return an Integer representing the difference between self and other
    :raises ValueError if other is not of type Integer or if they are not in the same base
    '''
    if not isinstance(other, Integer):
        raise ValueError("Error - Subtraction on two different objects.")
    if self.getBase() != other.getBase():
        raise ValueError("Error - Subtraction on two numbers with different bases.")
    new = Integer(self._base, repr(self))
    t = 0
    for i in range(len(new) - len(other)):
        other.append(0)

    for i in range(0, len(self)):
        new[i] = new[i] - (other[i] + t)
        if new[i] < 0:
            t = 1
        else:
            t = 0
        if t:
            new[i] += new.getBase()

    new.clearIsb()
    return new
```

The algorithm is very similar to the addition, we implement it like we would do it on the paper, taking advantage of how we keep the digits in the Integer class. If our number is say 12345 is base 10, then the array of digits will be [5, 4, 3, 2, 1].

For the subtraction of 54321 and 50111 we will do the following

54321 -> the digits are [1, 2, 3, 4, 5]

50111 -> the digits are [1, 1, 1, 0, 5]

subtracting we get:

[0, 1, 2, 4, 0]

Then we have to eliminate the leading zeroes, getting the following digits array:

[0, 1, 2, 4] in base 10, corresponding to 4210, which is the result of the subtraction.

Multiplication

```
def __mul__(self, other):
    '''
    Function to implement the multiplication of an Integer with another Integer
    :param other - the Multiplier (the Multiplicand is self)
    :return an Integer representing the product of self and other
    :raises ValueError if other is not of type Integer or if they are not in the same base
    '''
    if not isinstance(other, Integer):
        raise ValueError("Error - Multiplication only allowed between Integers.")
    if self.getBase() != other.getBase():
        raise ValueError("Error - Multiplication on Integers with different bases.")
    new = Integer(self._base, "0" * (len(self) + len(other) - 1))
    t = 0
    for i in range(0, len(self)):
            for j in range(0, len(other)):
                new[i + j] += self[i] * other[j]
    for i in range(0, len(new)):
        new[i] += t
        t = new[i] // self.getBase()
        new[i] = new[i] % self.getBase()

    while t > 0:
        new.append(t % self._base)
        t = t // self._base

    return new
```

The algorithm has complexity of O(N * M) where N is the number of digits in the first number and M is the number of digits in the second number.

Floor division (returns the quotient)

```
def __floordiv__(self, other):
    '''
    Function to implement the division of an Integer with a one digit Integer
    :param other - the Divisor (the Dividend is self)
    :return an Integer representing the quotient of the division of self and other
    :raises ValueError if other is not a one digit integer
    '''
    if not isinstance(other, int):
        raise ValueError("Error - Can only divide with small integers.")
    r = 0
    new = Integer(self.getBase(), repr(self))
    for i in reversed(range(0, len(new))):
        r = new.getBase() * r + new[i]
        new[i] = r // other
        r = r % other
    new.clearIsb()
    return new
```

Modulo operator - % (returns the remainder)

```
def __mod__(self, other):
    '''
    Function to implement the division of an Integer with a one digit Integer
    :param other - the Divisor (the Dividend is self)
    :return an Integer representing the remainder of the division of self and other
    :raises ValueError if other is not a one digit integer
    '''
    if not isinstance(other, int):
        raise ValueError("Error - Modulo only defined on integers.")
    r = 0
    for i in reversed(range(0, len(self))):
        r = (r * self.getBase() + self[i]) % other
    return r
```

Author: Rusu Cosmin

Comparator of two Integer objects

```
def _compare(self, other):
    '''
    Comparator function for the Integer class.
    :param other: an Integer
    :return: 1 if self > other
             0 if self == other
            -1 if self < other
    '''
    self.clearlsb()
    other.clearlsb()
    if len(self) > len(other):
        return 1
    elif len(self) < len(other):
        return -1
    else:
        for i in reversed(range(len(self))):
            if self[i] > other[i]:
                return 1
            elif self[i] < other[i]:
                return -1
        return 0

def __eq__(self, other):
    #== operator on self and other
    return self._compare(other) == 0

def __lt__(self, other):
    #< operator on self and other
    return self._compare(other) == -1

def __le__(self, other):
    # <= operator on self and other
    return self._compare(other) <= 0

def __gt__(self, other):
    # operator on self and other
    return self._compare(other) == 1

def __ge__(self, other):
    # >= operator on self and other
    return self._compare(other) >= 0
```

Author: Rusu Cosmin

Work T3

Implement the algorithms for conversion between two bases

Substitution Method

```
def substitutionMethod(self, destBase):
    '''
    Function to convert self in another base using the Substitution Method (recommended when the
        source base is less than the destination base)
    :param destBase: an integer from the set [2, 3, 4, 5, 6, 7, 8, 9, 10, 16] representing the base we want to
        convert our number
    :return: an Integer representing self in the destination base
    '''
    destNumber = Integer(destBase, "0")
    power = Integer(destBase, "1")
    for i in range(len(self)):
        destNumber = destNumber + power * Integer(destBase, Integer.Symbols[self[i]])
        power = power * Integer(destBase, Integer.Symbols[self._base])
    return destNumber
```

Let $N_{(b)} = (a_m a_{m-1} \dots a_1 a_0, a_{-1} \dots a_{-n})_{(b)}$ be a real number in the source base b.

Substitution method:

- all the digits from the source representation are converted into the destination base: $(a_i)_{(b)} = (a'_i)_{(h)}, i = -n, \dots, -1, 0, \dots, m-1$
- Calculation performed in the destination base
- the base b is converted into base h: $b = (b')_{(h)}$
- we calculate in base h the following sum:

$$(N')_{(h)} = (a'_0)_{(h)} * (b')_{(h)}^{0} + (a'_1)_{(h)} * (b')_{(h)}^{1} + \dots + (a'_m)_{(h)} * (b')_{(h)}^{m} +$$
$$+ (a'_{-1})_{(h)} * (b')_{(h)}^{-1} + \dots + (a'_{-n})_{(h)} * (b')_{(h)}^{-n}$$

Note: The method is recommended for b<h, because: $(a_i)_{(b)} = (a'_i)_{(h)}, i = -n, \dots, -1, 0, \dots, m-1$, b= $b_{(h)}$, and we have to perform only multiplications/divisions by one digit.

The method of successive divisions/multiplications

```
def successiveDivison(self, destBase):
    '''
    Function to convert self in another base using the Successive Division Method (recommended when the
        source base is greater than the destination base)
    :param destBase: an integer from the set [2, 3, 4, 5, 6, 7, 8, 9, 10, 16] representing the base we want to
        convert our number
    :return: an Integer representing self in the destination base
    '''
    destNumber = Integer(destBase, "0")
    power = Integer(destBase, "1")
    aux = Integer(destBase, "10")
    copyOfSelf = deepcopy(self)
    while len(copyOfSelf) != 0:
        destNumber = destNumber + power * Integer(destBase, Integer.Symbols[copyOfSelf % destBase])
        copyOfSelf = copyOfSelf // destBase
        power = power * aux
    return destNumber
```

The method of successive divisions/multiplications:

- calculation in the source base
- b-source base and h-destination base
- keep dividing the first number and the quotient of the division, and take the remainders in reverse order

Note: The method is recommended for h<b, because we need to apply only divisions/multiplications by one digit.

Author: Rusu Cosmin

Rapid conversions

```
def rapidConversions(self, destBase):
  ret = ""
  representation = repr(self)
  representation = representation[::-1]
  if self._base < destBase:
    many = int(log(destBase, self._base))
    i = 0
    while i < self._length:
      curr = representation[i:i + many]
      curr = curr[::-1]
      digit = Integer(self._base, curr)
      digit = digit.substitutionMethod(destBase)
      ret += repr(digit)
      i += many
  else:
    many = int(log(self._base, destBase))
    for i in range(len(self)):
      curr = Integer(self._base, Integer.Symbols[self[i]])
      curr = curr.successiveDivison(destBase)
      for j in range(0, len(curr)):
        ret = ret + Integer.Symbols[curr[j]]
      for j in range(many - len(curr)):
        ret = ret + "0"
  return Integer(destBase, ret[::-1])
```

Rapid conversions conversions between bases which are powers of 2. Note that our algorithms works also for converting from base 3 to base 9 or vice versa (since 9 is a power of 3). This method is a generalized method for every base b and h, when b is a power of h or h is a power of b.

- Conversion from the source base $p=2^k$, p from the set $\{4=2^2,8=2^3,16=2^4\}$ into the destination base 2
  - Each digit from the source number in base $p=2^k$, the integer part and the fractional one, will be replaced by the corresponding group of k binary digits (adding if it is necessary insignificant zeros on the left) according to the rapid conversion table (or can be simple calculated since we have maximum k digits to convert - which is constant: O(1) time complexity)
- Conversion from the source base 2 into the destination base $q=2^k$, q from the set $\{4=2^2,8=2^3,16=2^4\}$
  - from right to left make groups of k binary digits (eventually we add on the left insignificant zeros to have a complete group)
  - the groups will be replaced by the corresponding digits in base $q=2^k$ according to the above rapid conversion table (or can be simple calculated since we have maximum k digits to convert- which is constant: O(1) time complexity)

Author: Rusu Cosmin

The method which uses an intermediate base

```
def intermediateBase(self, destBase):
  x = self.substitutionMethod(10)
  x = x.successiveDivison(destBase)
  return x
```

The method which uses an intermediate base

$N_{(b)} = N'_{(g)} = N''_{(h)}$

b - the source base

g – the intermediate base

h - the destination base

In particular, let g = 10 – calculus in base 10

- conversion from base b into base 10 – using the substitution method,
- conversion from base 10 into base h – using the method of successive divisions