Babes Bolyai University, Computer Science Major
Computational Logic, First Semester 2020-2021

Arithmetic operations and conversions application in Python

Susciuc Anastasia

01. December. 2020

1 Problem Statement

Implement an algorithm to convert positive numbers from a source base b to a destination base h (b, h 2, 3, . . . 10, 16). The conversion can be made by the substitution method, the successive divisions method, rapid conversions or conversions using an intermediate base. This algorithm must also perform basic arithmetic operations for positive integers (addition, subtraction, multiplication by digit, division by one digit).

2 Arithmetic Operations

2.1 Addition

The addition method receives as input 2 lists of digits (the digits that were introduced by the user) and the base we perform the operation in. It returns another list of digits that represent the result of the addition.

Parameters:
'base' - the base we perform the addition in
'num1' - the first term of the addition - a list consisting of the digits of the term in reverse order
'num2' - the second term of the addition - a list consisting of the digits of the term in reverse order

Algorithm 1 Addition Algorithm

```
Result: a list of digits in base 'base' that form the result of the addition
$carry \leftarrow 0$
for $i \leftarrow 0$ to $n$ do
    $digit_{10} \leftarrow (num_1[i] + num_2[i] + carry) \bmod base$
    $carry \leftarrow (num_1[i] + num_2[i] + carry)$ div $base$
    $result[i] \leftarrow digit_{10}$
end
if carry is not 0 then
    $result[n] \leftarrow carry$
end
```

2.2 Subtraction

The subtraction method receives as input 2 lists of digits (the digits that were introduced by the user) and the base we perform the operation in. It returns another list of digits that represent the result of the subtraction.

Parameters:
'base' - the base we perform the multiplication in
'num1' - the first term of the subtraction - a list consisting of the digits of the term in reverse order
'num2' - the second term of the subtraction - a list consisting of the digits of the term in reverse order

Algorithm 2 Subtraction Algorithm

```
Result: a list of digits in base 'base' that form the result of the subtraction
$borrow \leftarrow 0$
for $i \leftarrow 0$ to $n$ do
    $digit_{10} \leftarrow (num_1[i] - num_2[i] - borrow)$
    $borrow \leftarrow 1$
    if $digit_{10}$ smaller than 0 then
        $digit_{10} \leftarrow digit_{10} + base$
        $borrow \leftarrow 1$
    end
    $result[i] \leftarrow digit_{10}$
end
if borrow is not 0 then
    Can't perform the subtraction!
end
```

2.3 Multiplication by digit

The multiplication method receives as input a list of digits (forming the number introduced by the user), a single digit number and the base we perform the operation in. It returns another list of digits that represent the result of the multiplication.

Parameters:

'base' - the base we perform the multiplication in

'num1' - the first term of the multiplication - a list consisting of the digits of the term in reverse order

'num2' - a single-digit number in base 'base'

Algorithm 3 Multiplication Algorithm

```
Result: a list of digits in base 'base' that form the result of the multiplication
$carry \leftarrow 0$
for $i \leftarrow 0$ to $n$ do
    $digit_{10} \leftarrow (num_1[i] * num_2 + carry)$
    $carry \leftarrow digit_{10}\ div\ base$
    $digit_{10} \leftarrow digit_{10}\ mod\ base$
    $result[i] \leftarrow digit_{10}$
end
if $carry$ is not 0 then
    $result[n+1] \leftarrow carry$
end
```

2.4 Division by digit

The division method receives as input a list of digits (forming the number introduced by the user), a single digit number and the base we perform the operation in. It returns another list of digits that represent the result of the division.

Raises an error if the second number (the denominator) is zero.

Parameters:

'base' - the base we perform the division in

'num1' - the first term of the division - a list consisting of the digits of the term in reverse order

'num2' - a single-digit number in base 'base'

Algorithm 4 Division Algorithm

```
Result: a list of digits in base 'base' that form the result of the division
$remainder \leftarrow 0$
for $i \leftarrow n-1$ to 0 do
    $digit_{10} \leftarrow (remainder * base + num_1[i])$
    $result[i] \leftarrow digit_{10}\ div\ num_2$
    $remainder \leftarrow digit_{10}\ mod\ num_2$
end
```

3 Conversion Methods

3.1 Method of substitution

The substitution method receives as input two bases (the source base and the destination one) and a number (given as a list of digits). The output is represented by a list of digits in the destination base that form the initial number converted in the destination base. The destination base must be greater than the source base, otherwise an error is raised.

We compute the result by adding at every step 'i' the positional value (composed by the positional power multiplied by the digit 'i', multiplication calculated in the destination base). Then we need to update the positional power by multiplying in the destination base the power with the source base.

Parameters:
:param base1: a number from 2, 3, .., 10, 16
:param base2: a number from 2, 3, .., 10, 16
:param number: a list containing the digits of the given number in reverse order
:return: a list of digits that form when reversed the number 'number' in base 'base2'

Algorithm 5 Substitution Method

```
Result: a number in the destination base representing the converted initial number
$PositionalPower \leftarrow 1$
$result \leftarrow 0$
for $i \leftarrow 0$ to $n$ do
    $result += MultiplyInDestBase(PositionalPower, num[i])$
    $PositionalPower \leftarrow MultiplyInDestBase(PositionalPower, SourceBase)$
end
```

3.2 Method of successive divisions

The successive divisions method receives as input two bases (the source base and the destination one) and a number (given as a list of digits). The output is represented by a list of digits in the destination base that form the initial number converted in the destination base. The destination base must be smaller than the source base, otherwise an error is raised.

We compute the result step by step. While the initial number is not zero, we divide it by the destination base and the remainder is appended to the result. The converted number is in the reversed list of the remainders.

Parameters:
:param base1: a number from 2, 3, .., 10, 16
:param base2: a number from 2, 3, .., 10, 16
:param number: a list containing the digits of the given number in reverse order
:return: a list of digits that form the number 'number' in base 'base2'

Algorithm 6 Substitution Method

```
Result: a number in the destination base representing the converted initial number
$i \leftarrow 0$
while $num$ is not zero do
    $remainder \leftarrow ModuloInSourceBase($ number, DestBase $)$
    $num \leftarrow DivisionInSourceBase($ number, DestBase $)$
    $result[++i] \leftarrow remainder$
end
```

3.3 Method of using 10 as intermediate base

This method is based on the previous methods (Substitution and Successive Division Methods). Basically, two conversions are computed, one from the source base to the intermediary one and then, this result is converted to the destination source. This method is sometimes preferred because of human reasonings, as it is easier for us to use and to think in base 10.

3.4 Rapid Conversions

The Rapid Conversion is used when both the destination base and the source base are powers of 2 from the set 2, 4, 8, 16, otherwise an error is raised. The algorithm first converts the initial number to base 2 (by expanding every digit into 2, 3 or, respectively, 4 bits) and then to the destination base by taking groups of 2, 3 or respectively 4 bits and converting them to the destination base.

More precisely, when we convert from a base 2^p to base 2, we expand every digit in p bits. When we convert from base 2 into base 2^k,we take groups of k bits and we convert them into a digit from the destination base.

4 Implementation Considerations

The application is implemented using Object Oriented Programming and Layered Architecture. This way, it consists in 4 layers: the main layer(*main.py*), the user interface layer (*UIConsole.py*), the service layer (*service.py*) and the operations layer (*algorithms.py*).

main layer - starts the application and builds the objects

the user interface layer - prints the menu, interprets the user's commands, catches expected exceptions and calls the appropriate service method. Then it prints the result of the computations.

service layer - transforms the numbers from the users in lists of digits so that the calculations to be more easily performed. It raises errors in case that the user's number inputs are invalid. It also transforms the result of the computations (additions, subtractions, multiplications, divisions and conversions) into a number that can be printed on the console.

operations layer - here are implemented all the algorithms for operations and conversions.

The menu has 8 options the users can choose from. (0 - Exit the application, 1 - Add two numbers, 2 - Subtract two numbers, 3 - Multiply two numbers, 4 - Divide two numbers, 5 - Conversion using substitution method, 6 - Conversion using successive divisions, 7 - Conversion using rapid conversions, 8 - Conversion using 10 as intermediate base). The user should provide a valid command. The programme works only with positive integers numbers. After choosing an option, the user will be provided with new instructions so as to insert the necessary data for that command.

5 Test Data

5.1 Addition

5.1.1 Valid tests

1001001(2) + 1001101110(2) = 1010110111(2)

1002102(3) + 1001(3) = 1010110(3)

1891083792(10) + 983729(10) = 1892067521(10)
1ABDEF3792(16) + 9AF9(10) = 1ABDEFD28B(16)
65261(8) + 0(8) = 65261(8)
65(8) + 66(8) = 153(8)

5.1.2 Invalid tests

1891083792(13) + 983729(13) - 'Invalid base!' (the calculator only performs computations for numbers that are written in basis from 2, 3, ..., 10, 16)
1891083792(7) + 983729(7) - 'All digits must be smaller than the base!' (all number's digits must be strictly smaller than the base)
[-5161](7) + 5(7) - 'Invalid input!' (only positive integers are allowed)

5.2 Subtraction

5.2.1 Valid tests

1001001(2) - 1010(2) = 111111(2)
726516(8) - 55552(8) = 650744(8)
9999999(10) - 0(10) = 9999999(10)
9999999(16) - 0(16) = 9999999(16)
ABE45(16) - 865F(16) = A37E6(16)
120421(5) - 1111(5) = 114310(5)

5.2.2 Invalid tests

1891083792(13) - 983729(13) :- 'Invalid base!' (the calculator only performs computations for numbers that are written in basis from 2, 3, ..., 10, 16)
1891083792(7) - 983729(7) :- 'All digits must be smaller than the base!' (all number's digits must be strictly smaller than the base)
[-5161](7) - 5(7) :- 'Invalid input!' (only positive integers are allowed)
625(7) - 112236(7) :- 'Cannot subtract numbers, the second is greater than the first!' (the result has to be a positive number)

5.3 Multiplication with one digit

5.3.1 Valid tests

1001001(2) * 0(2) = 0(2)
1001001(2) * 1(2) = 1001001(2)
1003241(5) * 4(5) = 4024114(5)
652625610(7) * 6(7) = 5543330160(7)
8700912947(10) * 6(10) = 52205477682(10)
AB5672FE(16) * A(16) = 6B1607DEC(16)

5.3.2 Invalid tests

189(18) * 8(18) - ’Invalid base!’ (the calculator only performs computations for numbers that are written in basis from 2, 3, ..., 10, 16)

1891083792(7) * 983729(7) - ’All digits must be smaller than the base!’ (all number’s digits must be strictly smaller than the base)

[-5161](7) * 5(7) - ’Invalid input!’ (only positive integers are allowed)

5161(8) * 57(8) - ’second number should be a single digit number’

5161(8) * J(8) - ’second number is invalid!’

5.4 Division

5.4.1 Valid tests

1001001(2) : 1(2) = 1001001(2) remainder 0(2)

652651(7) : 5(7) = 123251(7) remainder 3(7)

98153(10) : 6(10) = 16358(10) remainder 5(10)

98A1F5C3(16) : 7(16) = 15CDFE89(16) remainder 4(16)

5.4.2 Invalid tests

[-5161](7) * 5(7) - ’Invalid input!’ (only positive integers are allowed)

189(18) : 8(18) - ’Invalid base!’ (the calculator only performs computations for numbers that are written in basis from 2, 3, ..., 10, 16)

5161(8) : 57(8) - ’second number should be a single digit number’

5161(8) : J(8) - ’second number is invalid!’

5161(8) : 0(8) - ’can’t divide by zero!’

5.5 Substitution method

5.5.1 Valid tests

32410(5) = 3047(9)

71682817(9) = 20D3988(16)

82711(9) = 82711(9)

5.5.2 Invalid tests

82711(9) = ? (7) - ’The base to convert the number should be smaller than the initial base!’

82711(9) = ?(J) - ”Second base is invalid!”

82711(H) = ?(8) - ”First base is invalid!”

5.6 Succesive Divisions

5.6.1 Valid tests

32410(9) = 1141344(5)

A677BF(16) = 10909631(10)

322110(4) = 12010020(3)

5.6.2 Invalid tests

51511(6) = ?(7) - ”The base to convert the number should be greater than the initial base!”
82711(9) = ?(H) ”Second base is invalid!”
82711(J) = ?(7) ”First base is invalid!”

5.7 Intermediate Base

5.7.1 Valid tests

32110(4) = 1110010100(2)
ABCDE(16) = 703710(10)
9991728(10) = 212013120300(4)
1210201(4) = 1210201(4)
41310(5) = A91(16)

5.7.2 Invalid tests

82711(J) = ?(8) - ”First base is invalid!”
82711(10) = ?(J) - ”Second base is invalid!”

5.8 Rapid Conversions

5.8.1 Valid tests

32110(4) = 1110010100(2)
AF579E(16) = 53653636(8)
53653636(8) = AF579E(16)
101100111001(2) = 5471(8)
111(2) = 14(4)

5.8.2 Invalid tests

82711(J) = ?(8) - ”First base is invalid!”
82711(16) = ?(J) - ”Second base is invalid!”
52541(6) = ?(8) - ”The base should be a valid power of 2”
82711(16) = ?(5) - ”The base should be a valid power of 2”

6 Conclusion

This application can be easily used for computing operations and conversions on positive integers written in any base from 2, 3, ..., 10, 16. Implemented in Python, using OOP, this applications combines two important notions that a Computer Science student learned in his/her first semester of studies.