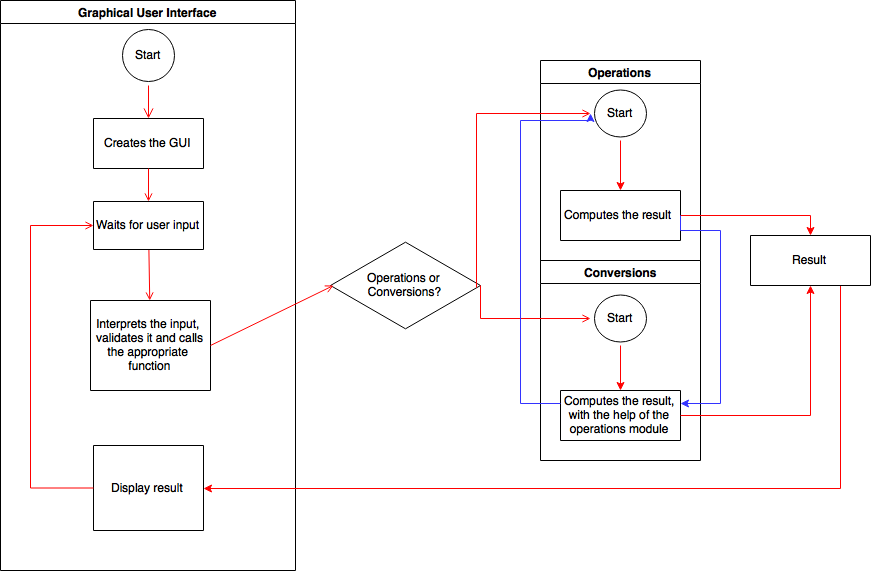
1. **Problem statement**

The application must implement algorithms for:

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

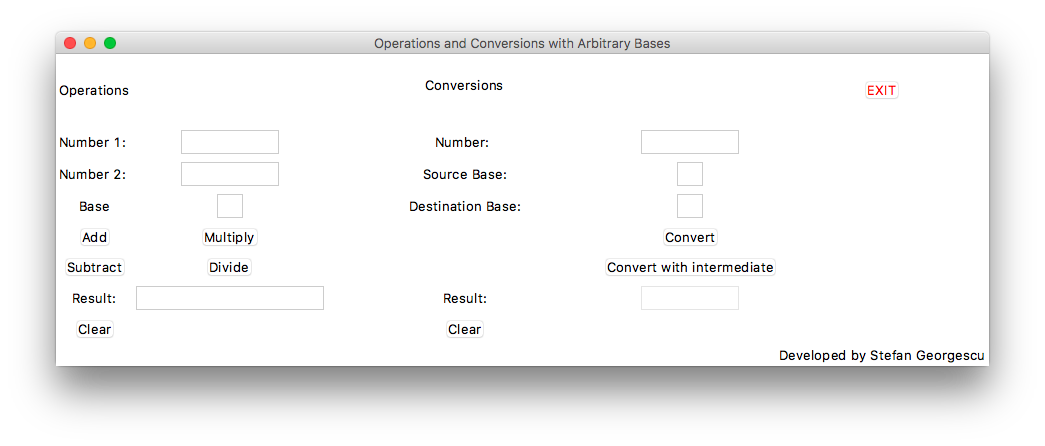
and must have a menu such that all operations and conversion methods to be verified separately.

1. **Sub-algorithm diagram**



1. **User Interface Walkthrough**

In the following section a short guide will be provided to ensure that the app is used at it’s maximum competence.



Screenshot of the home screen (the only one)

In the left hand side, we have the operations module. The user has to input two numbers, one in each field, and also enter a base of the two numbers.

The add button and the subtract button perform the addition or subtraction of the two numbers.

The multiply or divide buttons will perform the correspondent operations only if the second number is formed of one digit, as a consequence of the problem statement.

The result will be placed below, where it can be accessed (selected and copied) but not modified.

In the middle we have the conversions module. The user has to input one number and two bases, the source base and the destination base.

If the convert button is pressed, the program will decide which conversion method suits the situation and will apply the corresponding algorithm.

If the convert with intermediate button is pressed, the algorithm will convert the number with the intermediate base 10, and will show a message box to prove that, by displaying the number in it’s decimal interpretation.

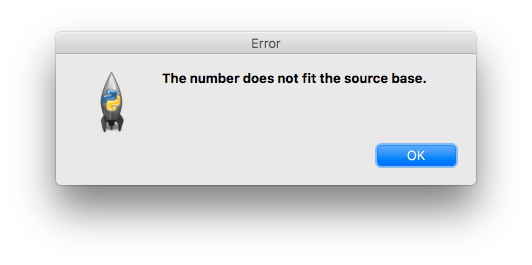
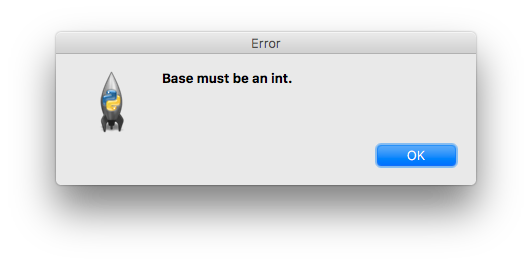
The result will behave as previous.

The two clear buttons delete the data in all fields for their corresponding module.

The exit button exits the app.

1. **Used data type specification**

The program takes from the user interface all numbers and bases as strings. It will do data validation before pushing them to the back end, and will give errors if something is not right.



1. **Specifications for the methods used**

As stated in a comment in the source code, which will be available at the end of this document, I will not provide documentation for methods regarding the graphical user interface, as they are not of use. In what follows we will look at functions from the operations and the conversions module.

**Operations**

**def get\_decimal\_value**(digit):  
 *"""  
 Returns the decimal value of a digit in any base.* ***:param*** *digit: the digit as a string* ***:return****: the value as an int  
 """*

**def get\_string\_value**(digit):  
 *"""  
 Returns the hexadecimal value of a digit in any base* ***:param*** *digit: the int value of a digit* ***:return****: the string with the hexadecimal value  
 """*

**def add\_digits**(digit1, digit2, carry, base):  
 *"""  
 Adds two digits received in any base and provides the next carry as an int  
 and the digit that must be added to the result as a string* ***:param*** *digit1: one digit, as a string* ***:param*** *digit2: the other digit, as a string* ***:param*** *carry: the previous carry* ***:param*** *base: the base in which we are doing the calculation* ***:return****: as mentioned above  
 """*

**def addition**(number1, number2, base):  
 *"""  
 Gets 2 numbers in the same base as a string and returns the result as a string* ***:param*** *number1: the first number as a string* ***:param*** *number2: the second number as a string* ***:param*** *base: the base of the two numbers* ***:return****: the result as a string  
 """*

**def subtract\_digits**(digit1, digit2, transport, base):  
 *"""  
 Gets the digits of the number where digit1 is from the minuend and digit2 is from the subtrahend* ***:param*** *digit1: a string containing one digit* ***:param*** *digit2: a string containing one digit* ***:param*** *transport: 0 or -1 as int* ***:param*** *base: the base* ***:return****: the carry as an int and the digit to be added as a string  
 """*

**def subtraction**(number1, number2, base):  
 *"""  
 Performs the difference of two numbers, where number 1 is the minuend and number 2 is the subtrahend,  
 number2 <= number1* ***:param*** *number1: a string* ***:param*** *number2: a string* ***:param*** *base: an int* ***:return****: the result of the subtraction  
 """*

**def multiply\_digits**(digit1, digit2, carry, base):  
 *"""  
 Generates the carry and the result from the multiplication of two digits* ***:param*** *digit1: a string containing one digit* ***:param*** *digit2: a string containing one digit* ***:param*** *carry: an int* ***:param*** *base: an int* ***:return****: the carry as an int and the digit to be added as a string  
 """*

**def multiply**(number1, number2, base):  
 *"""  
 Performs the multiplication of two numbers, where number 2 has only one digit* ***:param*** *number1: string* ***:param*** *number2: string* ***:param*** *base: an int* ***:return****: the result of the operation  
 """*

**def divide\_digits**(digit1, digit2, remainder, base):  
 *"""  
 Provides the result of dividing two digits, digit1 is divided by digit2* ***:param*** *digit1: a string containing one digit* ***:param*** *digit2: a string containing one digit* ***:param*** *remainder: the remainder, an int* ***:param*** *base: the base, an int* ***:return****: the next remainder as an int and the digit that has to be added as string.  
 """*

**def division**(number1, number2, base):  
 *"""  
 Performs the division of two numbers, number 1 is divided by number 2, where number2 is only one digit* ***:param*** *number1: a string* ***:param*** *number2: a string* ***:param*** *base: an int* ***:return****: the quotient and the remainder  
 """*

**Conversions**

**def substitution\_method**(number, source\_base, destination\_base):  
 *"""  
 This is used when source\_base < destination\_base* ***:param*** *number: a string* ***:param*** *source\_base: int* ***:param*** *destination\_base: int* ***:return****: the converted value  
 """*

**def successive\_divisions**(number, source\_base, destination\_base):  
 *"""  
 This is used when source\_base > destination\_base* ***:param*** *number: a string* ***:param*** *source\_base: an int* ***:param*** *destination\_base: an int* ***:return****: the result  
 """*

**def intermediate\_base**(number, source\_base, destination\_base):  
 *"""  
 Will compute the conversion using 10 as an intermediate base, using the proper conversion.* ***:param*** *number: a string* ***:param*** *source\_base: an int* ***:param*** *destination\_base: an int* ***:return****: the result in base 10 and the result in the destination base  
 """*

**def rapid\_conversions**(number, source\_base, destination\_base):  
 *"""  
 Converts the number using the rapid conversions method* ***:param*** *number: a string* ***:param*** *source\_base: an int, power of 2* ***:param*** *destination\_base: an int, power of 2* ***:return****: the result  
 """*

NB! Where the type of the result is not mentioned, it is of type string.

1. **Test Data**
2. AB650(16) = 2533120(8)
3. 2013(4) = 343(6)
4. 222(4) = 11011110(2)
5. 1735(8) = 989(10)
6. 23045(6) + 100254(6) = 123343(6)
7. ABCDE(16) + D9037F(16) = E3C05D(16)
8. 102387(9) – 64502(9) = 26785(9)
9. 102003(4) – 3333(4) = 32010(4)
10. 7023(8) \* 5(8) = 43137(8)
11. A23F4(16) \* B(16) = 6F8B7C(16)
12. 120456(8) : 6(8) = 13421(8) remainder 0(8)
13. 2A0F86(16) : E(16) = 3011B(16) remainder C(16)