Main:

*"""  
Developed by Stefan in 5-12-2016  
All copyrights reserved  
"""***from** ui.gui **import** GUI  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 app = GUI()  
 app.run\_app()

Graphical User Interface:

**import** traceback  
  
**from** tkinter **import** \*  
**from** tkinter **import** messagebox  
  
**from** domain.conversions **import** substitution\_method, rapid\_conversions, successive\_divisions, intermediate\_base  
**from** domain.operations **import** addition, subtraction, multiply, division, get\_decimal\_value  
**from** domain.util **import** strip\_insignificant\_zeros  
  
"""  
This part of the code only represents the graphical user interface and is completely independent from   
the code required to solve the tasks. As a consequence, I will only comment the methods that are relevant  
to the problem statement.   
"""  
  
  
**class GUI**:  
 **def** \_\_init\_\_(self):  
 self.master = Tk()  
 self.master.title("Operations and Conversions with Arbitrary Bases")  
  
 self.operation\_title = Label(self.master, text="Operations")  
 self.operation\_title.grid(row=0, column=0)  
 self.number1\_label = Label(self.master, text="Number 1: ")  
 self.number1\_label.grid(row=1, column=0, pady=5)  
 self.number1\_field = Entry(self.master, width=10)  
 self.number1\_field.grid(row=1, column=1)  
 self.number2\_label = Label(self.master, text="Number 2: ")  
 self.number2\_label.grid(row=2, column=0, pady=5)  
 self.number2\_field = Entry(self.master, width=10)  
 self.number2\_field.grid(row=2, column=1)  
 self.base\_label = Label(self.master, text="Base")  
 self.base\_label.grid(row=3, column=0, pady=5)  
 self.base\_selector = Entry(self.master, width=2)  
 self.base\_selector.grid(row=3, column=1)  
 self.addition\_button = Button(self.master, text="Add", command=self.addition\_button\_pressed)  
 self.addition\_button.grid(row=4, column=0, pady=5)  
 self.subtraction\_button = Button(self.master, text="Subtract", command=self.subtraction\_button\_pressed)  
 self.subtraction\_button.grid(row=5, column=0, pady=5)  
 self.multiply\_button = Button(self.master, text="Multiply", command=self.multiply\_button\_pressed)  
 self.multiply\_button.grid(row=4, column=1)  
 self.divide\_button = Button(self.master, text="Divide", command=self.divide\_button\_pressed)  
 self.divide\_button.grid(row=5, column=1)  
 self.result\_operations\_label = Label(self.master, text="Result:")  
 self.result\_operations\_label.grid(row=6, column=0, pady=5)  
 self.result\_operations\_field = Entry(self.master, width=20)  
 self.result\_operations\_field.grid(row=6, column=1)  
  
 self.clear\_operations\_button = Button(self.master, text="Clear", command=self.clear\_operations\_button\_pressed)  
 self.clear\_operations\_button.grid(row=7, column=0, pady=5)  
  
 self.conversions\_title = Label(self.master, text="Conversions")  
 self.conversions\_title.grid(row=0, column=2, pady=(20, 30))  
 self.number\_label = Label(self.master, text="Number: ")  
 self.number\_label.grid(row=1, column=2)  
 self.number\_field = Entry(self.master, width=10)  
 self.number\_field.grid(row=1, column=3)  
 self.source\_base\_label = Label(self.master, text="Source Base:")  
 self.source\_base\_label.grid(row=2, column=2)  
 self.source\_base\_selector = Entry(self.master, width=2)  
 self.source\_base\_selector.grid(row=2, column=3)  
 self.destination\_base\_label = Label(self.master, text="Destination Base:")  
 self.destination\_base\_label.grid(row=3, column=2, padx=80)  
 self.destination\_base\_selector = Entry(self.master, width=2)  
 self.destination\_base\_selector.grid(row=3, column=3)  
 self.convert\_button = Button(self.master, text="Convert", command=self.convert\_button\_pressed)  
 self.convert\_button.grid(row=4, column=3)  
 self.convert\_with\_intermediate\_button = Button(self.master, text="Convert with intermediate", command=self.convert\_with\_intermediate\_button\_pressed)  
 self.convert\_with\_intermediate\_button.grid(row=5, column=3)  
 self.result\_conversions\_label = Label(self.master, text="Result:")  
 self.result\_conversions\_label.grid(row=6, column=2)  
 self.result\_conversions\_field = Entry(self.master, width=10)  
 self.result\_conversions\_field.grid(row=6, column=3)  
 self.result\_conversions\_field.configure(state=DISABLED)  
  
 self.clear\_conversions\_button = Button(self.master, text="Clear", command=self.clear\_conversions\_button\_pressed)  
 self.clear\_conversions\_button.grid(row=7, column=2)  
   
 self.author\_label = Label(self.master, text="Developed by Stefan Georgescu")  
 self.author\_label.grid(row=8, column=4)  
 self.exit\_button = Button(self.master, text="EXIT", foreground='red', command=self.exit\_button\_pressed)  
 self.exit\_button.grid(row=0, column=4)  
  
 **def \_\_get\_operations\_data**(self):  
 **return** self.number1\_field.get(), self.number2\_field.get(), self.base\_selector.get()  
  
 **def \_\_perform\_conversion**(self, method, number, source\_base, destination\_base):  
 self.result\_conversions\_field.configure(state='normal')  
 self.result\_conversions\_field.delete(0, len(self.result\_conversions\_field.get()))  
 self.result\_conversions\_field.insert(END, strip\_insignificant\_zeros(method(number, source\_base, destination\_base)))  
 self.result\_conversions\_field.configure(state='readonly')  
  
 **def \_\_perform\_operation**(self, operation, number1, number2, base):  
 self.result\_operations\_field.configure(state='normal')  
 self.result\_operations\_field.delete(0, len(self.result\_operations\_field.get()))  
 self.result\_operations\_field.insert(END, strip\_insignificant\_zeros(operation(number1, number2, base)))  
 self.result\_operations\_field.configure(state='readonly')  
   
 **def addition\_button\_pressed**(self):  
 number1, number2, base = self.\_\_get\_operations\_data()  
 **try**:  
 base = int(base)  
 **if not** self.\_\_base\_validator(base):  
 messagebox.showinfo("Error", "This program can only work with bases 2 to 10 and 16.")  
 **else**:  
 **if not** self.\_\_fits\_base(number1, base) **or not** self.\_\_fits\_base(number2, base):  
 messagebox.showinfo("Error", "One of the numbers does not fit the base.")  
 **else**:  
 self.\_\_perform\_operation(addition, number1, number2, base)  
 **except** ValueError:  
 messagebox.showinfo("Error", "Base must be an int.")  
  
 **def subtraction\_button\_pressed**(self):  
 number1, number2, base = self.\_\_get\_operations\_data()  
 **try**:  
 base = int(base)  
 **if not** self.\_\_base\_validator(base):  
 messagebox.showinfo("Error", "This program can only work with bases 2 to 10 and 16.")  
 **else**:  
 **if not** self.\_\_fits\_base(number1, base) **or not** self.\_\_fits\_base(number2, base):  
 messagebox.showinfo("Error", "One of the numbers does not fit the base.")  
 **else**:  
 self.\_\_perform\_operation(subtraction, number1, number2, base)  
 **except** ValueError:  
 messagebox.showinfo("Error", "Base must be an int.")  
   
 **def multiply\_button\_pressed**(self):  
 number1, number2, base = self.\_\_get\_operations\_data()  
 **try**:  
 base = int(base)  
 **if not** self.\_\_base\_validator(base):  
 messagebox.showinfo("Error", "This program can only work with bases 2 to 10 and 16.")  
 **else**:  
 **if not** self.\_\_fits\_base(number1, base) **or not** self.\_\_fits\_base(number2, base):  
 messagebox.showinfo("Error", "One of the numbers does not fit the base.")  
 **else**:  
 **if** len(number2) != 1:  
 messagebox.showinfo("Error", "This program can only multiply by one digit.")  
 **else**:  
 self.\_\_perform\_operation(multiply, number1, number2, base)  
 **except** ValueError:  
 messagebox.showinfo("Error", "Base must be an int.")  
 traceback.print\_exc()  
  
 **def divide\_button\_pressed**(self):  
 number1, number2, base = self.\_\_get\_operations\_data()  
 **try**:  
 base = int(base)  
 **if not** self.\_\_base\_validator(base):  
 messagebox.showinfo("Error", "This program can only work with bases 2 to 10 and 16.")  
 **else**:  
 **if not** self.\_\_fits\_base(number1, base) **or not** self.\_\_fits\_base(number2, base):  
 messagebox.showinfo("Error", "One of the numbers does not fit the base.")  
 **else**:  
 **if** len(number2) != 1:  
 messagebox.showinfo("Error", "This program can only multiply by one digit.")  
 **else**:  
 self.result\_operations\_field.configure(state='normal')  
 self.result\_operations\_field.delete(0, len(self.result\_operations\_field.get()))  
 result, remainder = division(number1, number2, base)  
 self.result\_operations\_field.insert(END, result + ' remainder ' + remainder)  
 self.result\_operations\_field.configure(state='readonly')  
 **except** ValueError:  
 messagebox.showinfo("Error", "Base must be an int.")  
   
 **def clear\_operations\_button\_pressed**(self):  
 self.number1\_field.delete(0, len(self.number1\_field.get()))  
 self.number2\_field.delete(0, len(self.number2\_field.get()))  
 self.base\_selector.delete(0, len(self.base\_selector.get()))  
 self.result\_operations\_field.configure(state='normal')  
 self.result\_operations\_field.delete(0, len(self.result\_operations\_field.get()))  
 self.result\_operations\_field.configure(state='readonly')  
  
 **def convert\_button\_pressed**(self):  
 number = self.number\_field.get()  
 source\_base = self.source\_base\_selector.get()  
 destination\_base = self.destination\_base\_selector.get()  
 **try**:  
 source\_base = int(source\_base)  
 destination\_base = int(destination\_base)  
 **if not** self.\_\_base\_validator(source\_base) **or not** self.\_\_base\_validator(destination\_base):  
 messagebox.showinfo("Error", "This program can only work with bases 2 to 10 and 16.")  
 **else**:  
 **if not** self.\_\_fits\_base(number, source\_base):  
 messagebox.showinfo("Error", "The number does not fit the source base.")  
 **else**:  
 **if** self.\_\_is\_rapid(source\_base, destination\_base):  
 self.\_\_perform\_conversion(rapid\_conversions, number, source\_base, destination\_base)  
 messagebox.showinfo("Success", "The result has been computed using rapid conversions.")  
 **elif** source\_base < destination\_base:  
 self.\_\_perform\_conversion(substitution\_method, number, source\_base, destination\_base)  
 messagebox.showinfo("Success", "The result has been computed using the substitution method.")  
 **else**:  
 self.\_\_perform\_conversion(successive\_divisions, number, source\_base, destination\_base)  
 messagebox.showinfo("Success", "The result has been computed using successive divisions.")  
 **except** ValueError:  
 messagebox.showinfo("Error", "Base must be an int.")  
   
 **def convert\_with\_intermediate\_button\_pressed**(self):  
 number = self.number\_field.get()  
 source\_base = self.source\_base\_selector.get()  
 destination\_base = self.destination\_base\_selector.get()  
 **try**:  
 source\_base = int(source\_base)  
 destination\_base = int(destination\_base)  
 **if not** self.\_\_base\_validator(source\_base) **or not** self.\_\_base\_validator(destination\_base):  
 messagebox.showinfo("Error", "This program can only work with bases 2 to 10 and 16.")  
 **else**:  
 **if not** self.\_\_fits\_base(number, source\_base):  
 messagebox.showinfo("Error", "The number does not fit the source base.")  
 **else**:  
 mid\_result, final\_result = intermediate\_base(number, source\_base, destination\_base)  
 messagebox.showinfo("Success", "The result has been computed using 10 as an intermediate base and "  
 "the corresponding value is {0}".format(mid\_result))  
 self.result\_conversions\_field.configure(state='normal')  
 self.result\_conversions\_field.delete(0, len(self.result\_conversions\_field.get()))  
 self.result\_conversions\_field.insert(END, final\_result)  
 self.result\_conversions\_field.configure(state='readonly')  
 **except** ValueError:  
 messagebox.showinfo("Error", "Base must be an int.")  
   
 **def clear\_conversions\_button\_pressed**(self):  
 self.number\_field.delete(0, len(self.number\_field.get()))  
 self.source\_base\_selector.delete(0, len(self.source\_base\_selector.get()))  
 self.destination\_base\_selector.delete(0, len(self.destination\_base\_selector.get()))  
 self.result\_conversions\_field.configure(state='normal')  
 self.result\_conversions\_field.delete(0, len(self.result\_conversions\_field.get()))  
 self.result\_conversions\_field.configure(state='readonly')  
   
 **def exit\_button\_pressed**(self):  
 messagebox.showinfo("Credits", "Application developed by Stefan Georgescu, student at the Faculty of "  
 "Mathematics and Informatics, group number 913. \n \nAny unauthorised use of "  
 "this code will upset the developer. \n \nThank you!")  
 self.master.quit()  
  
 @staticmethod  
 **def run\_app**():  
 mainloop()  
  
 @staticmethod  
 **def \_\_fits\_base**(number, base):  
 *"""  
 Checks if a number can be a representation in a given base* ***:param*** *number: a string* ***:param*** *base: an int* ***:return****: True if all the digits are smaller than the base, false otherwise  
 """* **for** digit **in** number:  
 **if** get\_decimal\_value(digit) >= base:  
 **return False  
 return True** @staticmethod  
 **def \_\_base\_validator**(base):  
 *"""  
 Checks if a given base is valid for the problem statement* ***:param*** *base: an int* ***:return****: True or False  
 """* **if** base < 2:  
 **return False  
 if** base **in** range(11, 16):  
 **return False  
 if** base > 16:  
 **return False  
 return True** @staticmethod  
 **def \_\_is\_rapid**(base1, base2):  
 *"""  
 Checks if two bases are powers of two* ***:param*** *base1: an int* ***:param*** *base2: an int* ***:return****: True or False  
 """* powers\_of\_2 = [2, 4, 8, 16]  
 **if** base1 **not in** powers\_of\_2:  
 **return False  
 if** base2 **not in** powers\_of\_2:  
 **return False  
 return True**

Conversions:

**from** domain.operations **import** multiply, addition, division  
  
  
**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  
 """* number = number[::-1] # Compute the inverse  
 result = '0'  
 multiplier = '1' # This will be the number we multiply each digit with  
 **for** digit **in** number:  
 current\_digit\_result = multiply(multiplier, digit, destination\_base) # Get the value corresponding to the digit  
 result = addition(result, current\_digit\_result, destination\_base) # Add that value to the result  
 multiplier = multiply(multiplier, source\_base, destination\_base) # Get the next power of the multiplier  
 **return** str(result) # The result has been computed in the destination base, so we just return the number  
  
  
**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  
 """* result = ''  
 **while** number != '':  
 number, remainder = division(number, destination\_base, source\_base)  
 result += remainder # We work with strings, so this adds the new remainder to the result  
 **return** result[::-1] # Return the mirror of what we have computed  
  
  
**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  
 """* **if** source\_base < 10: # Choose the proper conversion from the source base to base 10  
 result\_base10 = substitution\_method(number, source\_base, 10)  
 **else**:  
 result\_base10 = successive\_divisions(number, source\_base, 10)  
  
 **if** 10 < destination\_base: # Choose the proper conversion from base 10 to the destination base  
 result = substitution\_method(result\_base10, 10, destination\_base)  
 **else**:  
 result = successive\_divisions(result\_base10, 10, destination\_base)  
  
 **return** result\_base10, result  
  
  
**def generate\_zero**(number):  
 *"""  
 Will be used for rapid conversions* ***:param*** *number: the number of 0 we need* ***:return****: a string made of 0s, exactly number times  
 """* result = ''  
 **for** i **in** range(0, number):  
 result += '0'  
 **return** result  
  
  
**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  
 """* power\_dict = {2: 1, 4: 2, 8: 3, 16: 4} # We store the possible bases and their corresponding exponent  
 power = power\_dict[source\_base] # Get the exponent of the source base  
 result = ""  
 # Generate the binary representation digit by digit  
 **for** digit **in** number:  
 # Get the binary representation of the current digit  
 current\_result = successive\_divisions(digit, source\_base, 2)  
 # Add 0s in front of the result if the representations is not on the required number of bits  
 **if** len(current\_result) != power:  
 current\_result = generate\_zero(power-len(current\_result)) + current\_result  
 result += current\_result  
  
 power = power\_dict[destination\_base] # Get the exponent of the destination base  
  
 # Check if the result we had from the previous part of the code has enough digits. If not, add 0s in front  
 **if** len(result) % power != 0:  
 result = generate\_zero(power - len(result) % power) + result  
  
 index = 0  
 result\_length = len(result)  
 final\_result = ''  
 **while** index < result\_length:  
 # Get the required number of digits (the exponent of the destination base)  
 temp = result[index:index + power]  
 # Convert the bits to the destination base  
 final\_result += substitution\_method(temp, 2, destination\_base)  
 # Move to the next group of digits  
 index += power  
 **return** final\_result

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  
 """* hexadecimal = {'A': 10, 'a': 10, 'B': 11, 'b': 11, 'C': 12, 'c': 12, 'D': 13, 'd': 13, 'E': 14, 'e': 14,  
 'F': 15, 'f': 15}  
 **try**:  
 digit = int(digit)  
 **return** digit  
 **except** ValueError:  
 **return** hexadecimal[digit]  
  
  
**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  
 """* decimal = {10: 'A', 11: 'B', 12: 'C', 13: 'D', 14: 'E', 15: 'F'}  
 **if** digit > 9:  
 **return** decimal[digit]  
 **return** str(digit)  
  
  
**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  
 """* digit1 = get\_decimal\_value(digit1)  
 digit2 = get\_decimal\_value(digit2)  
 **return** (digit1 + digit2 + carry) // base, get\_string\_value((digit1 + digit2 + carry) % base)  
  
  
**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  
 """* # Initialise all the required variables for the algorithm. This step will be present for all operations  
 number1 = number1[::-1] # This creates the mirror of the number, so we can go through the digits from  
 number2 = number2[::-1] # the end  
 result = ""  
 length\_number1 = len(number1)  
 length\_number2 = len(number2)  
 index = 0  
 carry = 0  
 # Will go through the digits one by one as long as there are digits available  
 **while** index < length\_number1 **and** index < length\_number2:  
 carry, digit\_to\_add = add\_digits(number1[index], number2[index], carry, base)  
 result += digit\_to\_add  
 index += 1  
  
 # This will execute if one of the numbers is longer than the other  
 **while** index < length\_number1:  
 carry, digit\_to\_add = add\_digits(number1[index], 0, carry, base)  
 result += digit\_to\_add  
 index += 1  
  
 **while** index < length\_number2:  
 carry, digit\_to\_add = add\_digits(number2[index], 0, carry, base)  
 result += digit\_to\_add  
 index += 1  
  
 # Finally, we check if we have a carry left so we can add it  
 **if** carry != 0:  
 result += get\_string\_value(carry)  
 **return** result[::-1]  
  
  
**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 and the digit to be added  
 """* digit1 = get\_decimal\_value(digit1)  
 digit2 = get\_decimal\_value(digit2)  
 **if** digit1 + transport < digit2:  
 **return** -1, get\_string\_value(digit1 + transport + base - digit2)  
 **else**:  
 **return** 0, get\_string\_value(digit1 + transport - digit2)  
  
  
**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  
 """* # Makes sure that the bigger number is the minuend  
 negative = **False** length\_number1 = len(number1)  
 length\_number2 = len(number2)  
 value\_number1 = int(number1, 16)  
 value\_number2 = int(number2, 16)  
 **if** value\_number1 < value\_number2:  
 number1, number2 = number2, number1  
 negative = **True** number1 = number1[::-1]  
 number2 = number2[::-1]  
 result = ""  
 index = 0  
 transport = 0  
  
 # The following snippet of code works similarly to the addition.  
 **while** index < length\_number1 **and** index < length\_number2:  
 transport, digit\_to\_add = subtract\_digits(number1[index], number2[index], transport, base)  
 result += digit\_to\_add  
 index += 1  
  
 **while** index < length\_number1:  
 transport, digit\_to\_add = subtract\_digits(number1[index], 0, transport, base)  
 result += digit\_to\_add  
 index += 1  
  
 **if** negative:  
 **return** "-" + result[::-1]  
 **return** result[::-1]  
  
  
**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 and the digit to be added  
 """* digit1 = get\_decimal\_value(digit1)  
 digit2 = get\_decimal\_value(digit2)  
 **return** (digit1 \* digit2 + carry) // base, get\_string\_value((digit1 \* digit2 + carry) % base)  
  
  
**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  
 """* number1 = number1[::-1]  
 result = ""  
 length\_number1 = len(number1)  
 index = 0  
 carry = 0  
 **while** index < length\_number1:  
 carry, digit\_to\_add = multiply\_digits(number1[index], number2, carry, base)  
 result += digit\_to\_add  
 index += 1  
 **if** carry != 0:  
 result += get\_string\_value(carry)  
 **return** result[::-1]  
  
  
**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 and the digit that has to be added.  
 """* digit1 = get\_decimal\_value(digit1)  
 digit2 = get\_decimal\_value(digit2)  
 **return** (remainder \* base + digit1) % digit2, get\_string\_value((remainder \* base + digit1) // digit2)  
  
  
**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  
 """* result = ""  
 length\_number1 = len(number1)  
 index = 0  
 remainder = 0  
 **while** index < length\_number1:  
 remainder, digit\_to\_add = divide\_digits(number1[index], number2, remainder, base)  
 **if** index != 0 **or** digit\_to\_add != '0': # Not adding 0 at the result if it would be the first digit of the result  
 result += digit\_to\_add  
 index += 1  
 **return** result, get\_string\_value(remainder)

Util:

**def strip\_insignificant\_zeros**(number):  
 *"""  
 Eliminates insignificant zeros from the beginning of a number* ***:param*** *number: the number as a string* ***:return****: the number without zeros  
 """* **if** all\_zero(number):  
 **return** '0'  
 **if** number[0] != "-":  
 **while** number[0] == '0' **and** number != '':  
 number = number[1:] # Eliminates the first digit of the number  
 **else**:  
 **while** number[1] == '0' **and** number != '-':  
 number = "-" + number[2:] # Eliminates the first digit of the number after the minus  
 **return** number  
  
  
**def all\_zero**(number):  
 *"""  
 Checks if a number is made up only of zeros.* ***:param*** *number: a string* ***:return****: True and False  
 """* **for** digit **in** number:  
 **if** digit != '0':  
 **return False  
 return True**