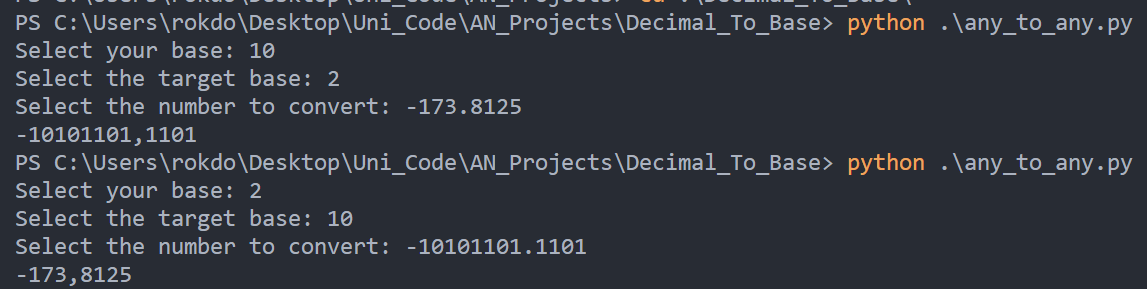
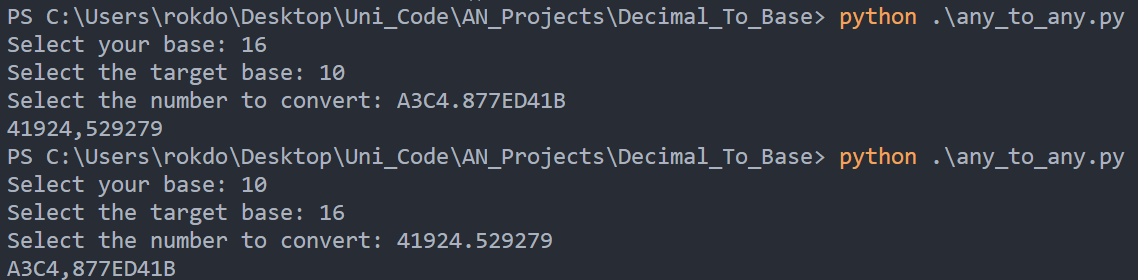
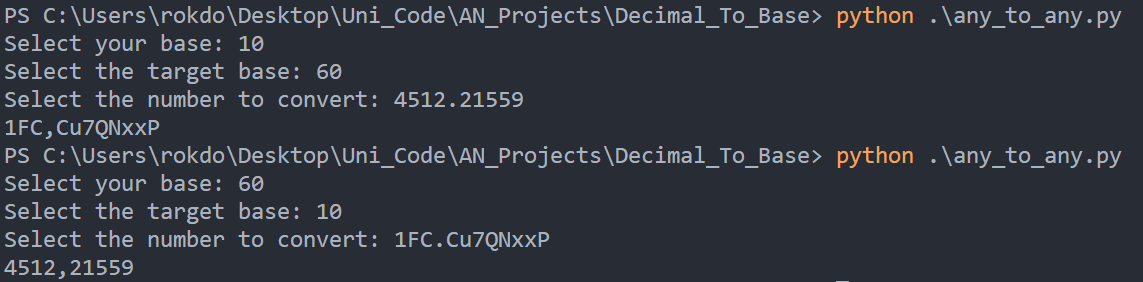
Elementos do grupo:

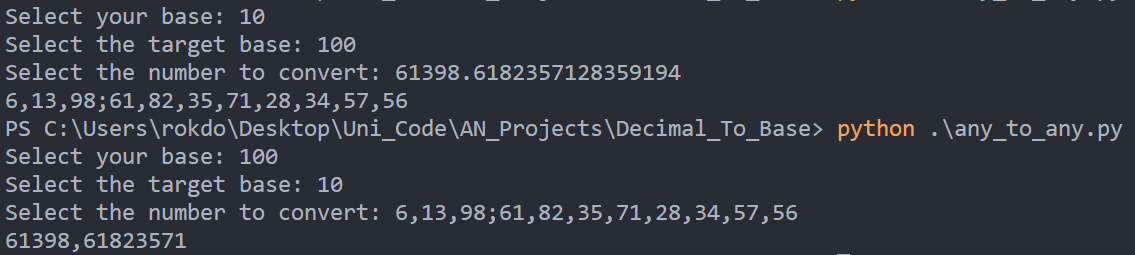
* Diogo Fonseca nº 79858
* Tomás Teodoro nº 80044
* Diogo Silva nº 79828
* Tiago Granja nº 79845

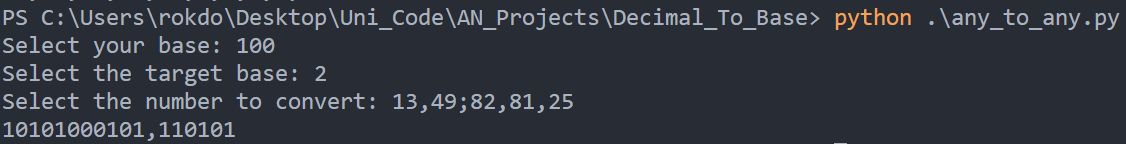
*Screenshots* do programa a funcionar. O programa é capaz de converter de **qualquer** base para **qualquer** base. É possível notar alguma imprecisão quando as bases tendem a ser muito grandes, sendo impossível anular o erro gerado pela imprecisão gerada quando o computador faz aritmética de *floating point* (embora tenha sido minimizada até certo ponto).

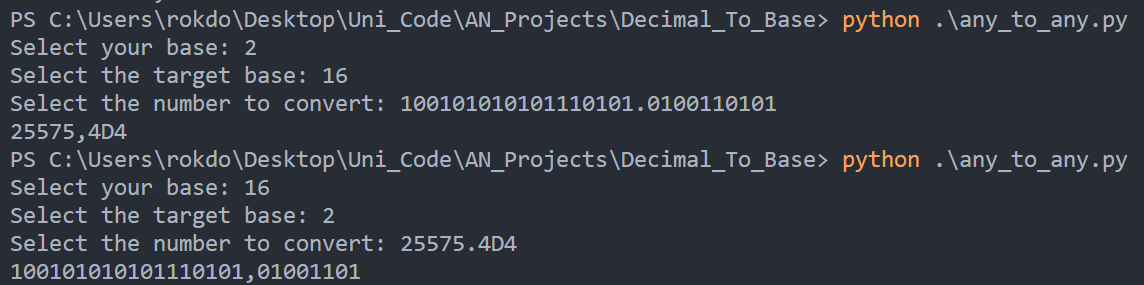


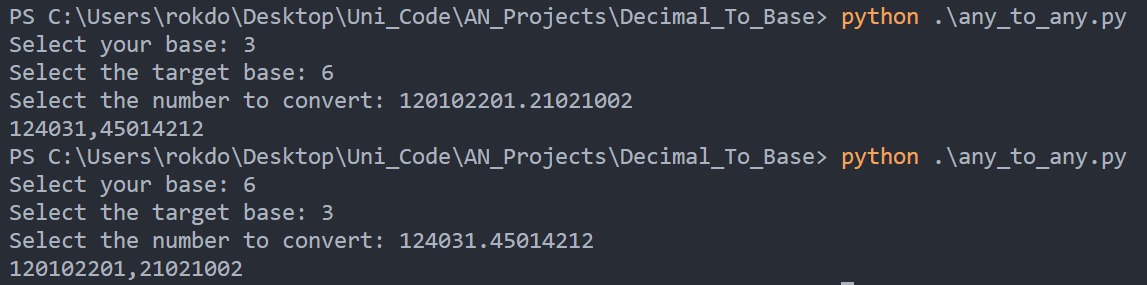












Código:

import math

def convert\_base(*x*, *base*):

    y = abs(*x*)

    if *base* <= 10:

        y = decimal\_to\_less\_than\_10(y, *base*)

        y = base\_round(y, *base*)

    elif *base* <= 61:

        y = decimal\_to\_less\_than\_62(y, *base*)

        y = base\_round(y, *base*)

    else:

        y = decimal\_to\_greater\_than\_61(y, *base*)

    if *x* < 0:

        y = "-" + y

    return y

def decimal\_to\_less\_than\_10(*x*, *base*):

    whole = decimal\_to\_less\_than\_10\_whole(*x*, *base*)

    fractionary = decimal\_to\_less\_than\_10\_fractionary(*x*, *base*)

    return whole + "," + fractionary

def decimal\_to\_less\_than\_10\_whole(*x*, *base*):

    y = str(int(*x*))

    if *x* >= *base*:

        y = str(decimal\_to\_less\_than\_10\_whole(*x* // *base*, *base*)) + str(int(*x*) % *base*)

    return y

def decimal\_to\_less\_than\_10\_fractionary(*x*, *base*):

    y = ""

    for i in range(9):

*x* -= int(*x*)

*x* = *x* \* *base*

        y += str(int(*x*))

    return y

def decimal\_to\_less\_than\_62(*x*, *base*):

    whole = decimal\_to\_less\_than\_62\_whole(*x*, *base*)

    fractionary = decimal\_to\_less\_than\_62\_fractionary(*x*, *base*)

    return whole + "," + fractionary

def decimal\_to\_less\_than\_62\_whole(*x*, *base*):

    y = str(int(*x*))

    if *x* >= *base*:

        y = str(decimal\_to\_less\_than\_62\_whole(*x* // *base*, *base*)) + int\_to\_alphanumeric(

            int(*x*) % *base*

        )

    else:

        y = int\_to\_alphanumeric(int(*x*))

    return y

def decimal\_to\_less\_than\_62\_fractionary(*x*, *base*):

    y = ""

    for i in range(9):

*x* -= int(*x*)

*x* = *x* \* *base*

        y += int\_to\_alphanumeric(int(*x*))

    return y

def decimal\_to\_greater\_than\_61(*x*, *base*):

    whole = decimal\_to\_greater\_than\_61\_whole(*x*, *base*)

    fractionary = decimal\_to\_greater\_than\_61\_fractionary(*x*, *base*)

    return whole + ";" + fractionary

def decimal\_to\_greater\_than\_61\_whole(*x*, *base*):

    y = str(int(*x*))

    if *x* >= *base*:

        y = (

            str(decimal\_to\_greater\_than\_61\_whole(*x* // *base*, *base*))

            + ","

            + str(int(*x*) % *base*)

        )

    return y

def decimal\_to\_greater\_than\_61\_fractionary(*x*, *base*):

    y = ""

*x* -= int(*x*)

*x* = *x* \* *base*

    y += str(int(*x*))

    for i in range(7):

*x* -= int(*x*)

*x* = *x* \* *base*

        y += "," + str(int(*x*))

    return y

def int\_to\_alphanumeric(*x*):

    if *x* >= 10 and *x* <= 35:

        return chr(*x* + 55)

    elif *x* >= 36 and *x* <= 61:

        return chr(*x* + 61)

    return str(*x*)

def alphanumeric\_to\_int(*char*):

    ascii\_value = ord(*char*)

    result = 0

    if ascii\_value >= 65 and ascii\_value <= 90:

        result = int(ascii\_value - 55)

    elif ascii\_value >= 97 and ascii\_value <= 122:

        result = int(ascii\_value - 61)

    else:

        result = int(*char*)

    return result

def base\_round(*x*, *base*):

    reverse = *x*[::-1]

    reverse\_list = list(reverse)

    reverse\_list.pop(0)

    if alphanumeric\_to\_int(reverse[0]) >= math.ceil(*base* / 2):

        reverse = reversed\_base\_string\_list\_add(reverse\_list, *base*)

    reverse\_list = remove\_zeros(reverse\_list)

    return "".join(reverse\_list)[::-1]

def reversed\_base\_string\_list\_add(*reverse\_list*, *base*):

    for i in range(len(*reverse\_list*)):

        if *reverse\_list*[i] == int\_to\_alphanumeric(*base* - 1):

*reverse\_list*[i] = "0"

        else:

*reverse\_list*[i] = chr(ord(*reverse\_list*[i]) + 1)

            break

    return *reverse\_list*

def remove\_zeros(*reverse\_list*):

    current\_char = *reverse\_list*[0]

    while current\_char == "0":

*reverse\_list*.pop(0)

        current\_char = *reverse\_list*[0]

    if *reverse\_list*[0] == ",":

*reverse\_list*.pop(0)

    return *reverse\_list*

*# ------------------- to decimal -------------------*

def convert\_to\_decimal(*number\_str*, *base*):

    number\_str\_list = [char for char in *number\_str*]

    result = ""

    prepend = ""

    if number\_str\_list[0] == "-":

        prepend = number\_str\_list.pop(0)

    if *base* <= 10:

        result = less\_than\_10\_to\_decimal(number\_str\_list, *base*)

    elif *base* <= 61:

        result = less\_than\_10\_to\_decimal(number\_str\_list, *base*)

    else:

        result = greater\_than\_61\_to\_decimal(number\_str\_list, *base*)

    return prepend + str(result)

def less\_than\_10\_to\_decimal(*number\_str\_list*, *base*):

    whole = less\_than\_61\_to\_decimal\_whole(*number\_str\_list*, *base*)

    fractionary = less\_than\_61\_to\_decimal\_fractionary(*number\_str\_list*, *base*)

    return whole + fractionary

def less\_than\_61\_to\_decimal\_whole(*number\_str\_list*, *base*):

    number\_list = []

    for character in *number\_str\_list*:

        if character == ".":

            break

        number\_list.append(alphanumeric\_to\_int(character))

    number\_list.reverse()

    k = 0

    result = 0

    for number in number\_list:

        result += number \* pow(*base*, k)

        k += 1

    return result

def less\_than\_61\_to\_decimal\_fractionary(*number\_str\_list*, *base*):

    result = 0.0

    if "." not in *number\_str\_list*:

        return result

    number\_list = []

    for character in *number\_str\_list*[::-1]:

        if character == ".":

            break

        number\_list.append(alphanumeric\_to\_int(character))

    number\_list.reverse()

    k = 1

    for number in number\_list:

        result += number / pow(*base*, k)

        k += 1

    return result

def greater\_than\_61\_to\_decimal(*number\_str\_list*, *base*):

    whole = greater\_than\_61\_to\_decimal\_whole(*number\_str\_list*, *base*)

    fractionary = greater\_than\_61\_to\_decimal\_fractionary(*number\_str\_list*, *base*)

    return whole + fractionary

def greater\_than\_61\_to\_decimal\_whole(*number\_str\_list*, *base*):

    number\_list = []

    buffer = ""

    for character in *number\_str\_list*:

        if character == ",":

            number\_list.append(int(buffer))

            buffer = ""

        elif character == ";":

            break

        else:

            buffer += character

    if buffer != "":

        number\_list.append(int(buffer))

    number\_list.reverse()

    k = 0

    result = 0

    for number in number\_list:

        result += number \* pow(*base*, k)

        k += 1

    return result

def greater\_than\_61\_to\_decimal\_fractionary(*number\_str\_list*, *base*):

    result = 0.0

    buffer = ""

    if ";" not in *number\_str\_list*:

        return result

    number\_list = []

    for character in *number\_str\_list*[::-1]:

        if character == ",":

            number\_list.append(int(buffer[::-1]))

            buffer = ""

        elif character == ";":

            break

        else:

            buffer += character

    if buffer != "":

        number\_list.append(int(buffer[::-1]))

    number\_list.reverse()

    k = 1

    for number in number\_list:

        result += number / pow(*base*, k)

        k += 1

    return result

base = int(input("Select your base: "))

target\_base = int(input("Select the target base: "))

number = input("Select the number to convert: ")

base\_10\_number = convert\_to\_decimal(number, base)

print(convert\_base(float(base\_10\_number), target\_base))