DOCUMENTAȚIE TEMĂ ELECTRONICĂ

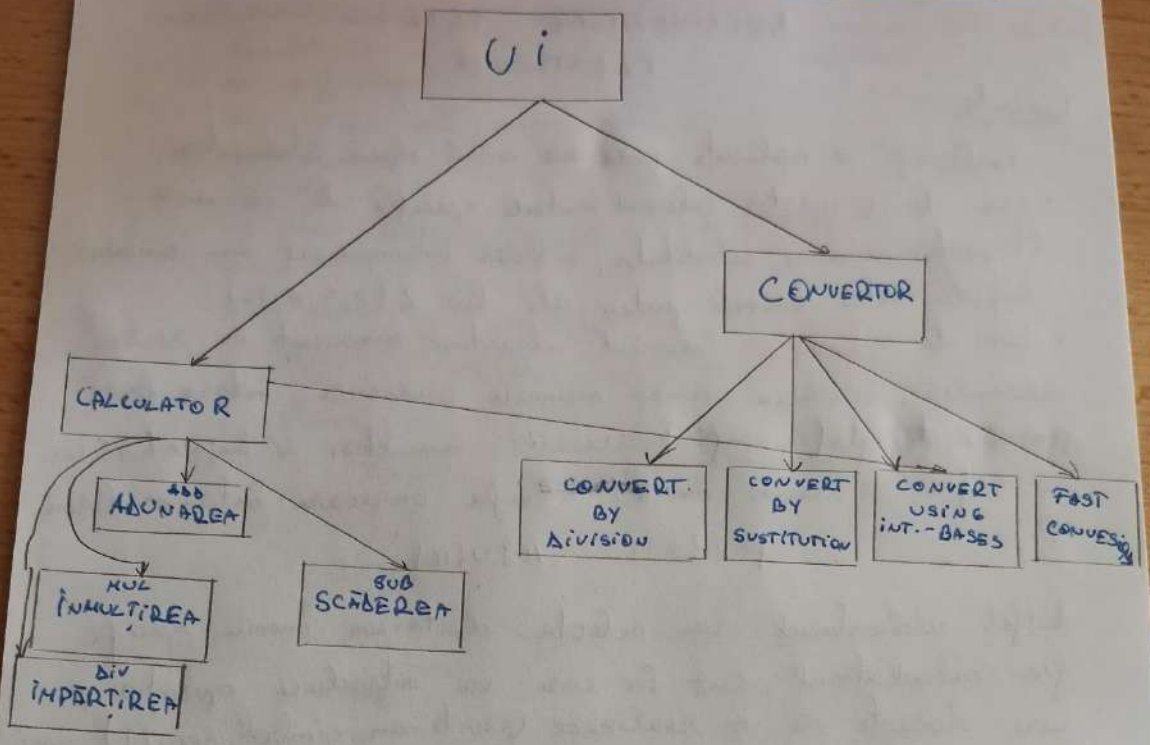
Cerință:

Realizați o aplicație care să aibă două întrebuințări:

- cea de convertor folosind metode specifice de conversie (împărțiri succesive, substituție, o bază intermediară sau conversii rapide între bazele puteri ale lui 2 (2, 4, 8, 16)
- cea de calculator folosind algoritmi cunoscuți va realiza adunarea, scăderea unor numere oarecare, într-o bază p aleasă de utilizator (nu neapărat aceeași), înmulțiri și împărțiri cu o cifră a unor numere și cifre oarecare într-o bază p.

$$p \in \{2, 3, \dots, 10\} \cup \{16\}$$

Astfel utilizatorul va selecta dintr-un meniu specific, fie calculatorul, caz în care va introduce operația pe care dorește să o realizeze (printr-un simbol specific), apoi se vor introduce operanzii și baza acestora, precum și baza în care dorim să se realizeze operația. Aplicația va returna rezultatul, fie convertorul, caz în care utilizatorul va selecta metoda prin care se va realiza conversia și va prelua un număr. Baza de numerație în care se află și baza de numerație în care vrem să îl convertim. Aplicația va returna numărul în baza destinație. Aplicația utilizează exclusiv numere naturale!



Tipuri de date folosite:

- tipul int disponibil în python
- tipul str disponibil în python
- tipul list disponibil în python
- clasa number ce permite stocarea unui număr și a bazei acestuia și realizarea operațiilor de adunare, scădere, înmulțire, împărțire și a conversiilor cu numerele stocate în ea

```
subalgoritm sub_ns(self, value) - scade două numere într-o bază
                                  arbitrară
    result = []
    bs = self.__baza
    ns1, ns2 = self.__ns, value
    ns1.reverse()
    ns2.reverse()
    lm = max(len(ns1), len(ns2))
    t = 0
    for index in range(lm):
        c1, c2 = get(ns1, ns2)
        dacă c1 - t < c2:
            result.append(bs + c1 - t - c2)
            t = 1
        altfel
            result.append(c1 - t - c2)
            t = 0
    result.elimină_0_res()
    result.reverse()
    dacă result == []:
        result = [0]
    dacă t == 1:
        aruncăm Eroare.
    return result
```

Date: self.__ns = ["1", "0", "2", "3", "8", "7"]
value = ["6", "4", "5", "0", "2"] bs = 9

Rezultate result = [2, 6, 7, 8, 5]

Precondiții: ambele numere stocate sub formă de listă de caract.

Postcondiții: rezultatul va fi ~~o~~ listă de numere în care pe fiecare poziție va fi o cifră a numărului

Subalgoritmi principali:

subalgoritm add_nr(self, value) - adună două numere în baza self.__base

```
result = []
bs = self._base
nr1 = self.__nr (în program am folosit deepcopy)
nr2 = value
nr1.reverse()
nr2.reverse()
lm = max(len(nr1), len(nr2))
t = 0
for index in range(lm):
    set(c1, c2) - se setează valorile
    dacă c1 + c2 + t < bs:
        result.append(c1 + c2 + t)
        t = 0
    altfel
        result.append((c1 + c2 + t) % bs)
        t = 1
dacă t == 1
    result.append(1)
result.reverse()
returnează result.
```

Date: self.__nr = ["2", "3", "0", "4", "5"] bs = 6
value = ["1", "0", "0", "2", "5", "4"]

Rezultate: rez = [1, 2, 3, 3, 4, 3]

Precondiții: Numerele vor fi stocate în liste prin forma de caract

Postcondiții: În rezultat se va genera o listă unde fiecare element a acestora rez. o altă.

```
subalgoritm div_by_digit (self, digit) - imparte cu o cifra
    result = []
    bs = self.__base
    nr = self.__nr
    ln = len(nr)
    t = 0
    dacă lungimea(digit) != 1:
        arunca eroare
    digit = get_digit(digit)
    for index in range(ln)
        c = get_digit(nr[index])
        result.append((bs*t+c)//digit)
        t = (bs*t+c)%digit
    elimină_0(result)
    return [result, t]
```

Date: self.__nr = ["1", "2", "0", "4", "5", "6"] bs = 8
digit = "6"

Rezultat: rez = [1, 5, 3, 3, 5]
rest = [0]

Preconditii: nr. stochat sub formă de obiect number
cifră sub formă de char

Postconditii: se returnează câtul și restul, câtul sub formă de listă, restul sub formă unui număr (integer)

```
subalgoritm mul_by_digit(self, digit) - înmulțirea unui nr. cu
    result = []                                  o cifră
    nr. = self.__nr
    nr.reverse
    lm = len(nr)
    t = 0
    atâta timp cât len(digit) != 1:
        aruncă Eroare
    digit = get_digit(digit)
    for index in range(lm):
        c = get_digit(nr[index])
        result.append((c*digit + t) % bs)
        t = (c*digit + t) // bs
    dacă t != 0:
        result.append(t)
    result.reverse()
    return result
```

Date: self.__nr = ["3", "2", "0", "0", "1", "B"] bs = 16

digit = "6"; Rez: rez = [6, 15, 8, 11, 7, 12]

Precondiții: Nr. stocate în clasa number și cifra ca char

Postcondiții: Se returnează o listă în care fiecare element reprezintă o cifră.

```
subalgoritm convert_by_substitution(self, new_base) - conversie prin substitutie
    daca new_base > self.__base atunci:
        base = self.__base
        cpy = self.__nr
        new_nr = []
        self.__nr = ['1']
        cpy.reverse()
        for index in range(len(cpy)):
            lst_power = self.__nr
            self.__nr = self.mul_by_number(cpy[index])
            new_nr.append(self.__nr)
            self.__nr = lst_power
            self.__nr = self.mul_by_number([base])
        self.__nr = new_nr[0]
        for index in range(1, len(new_nr)):
            self.__nr = [self.alt_digit(elem) for elem in
                         self.mul_by_number([base])]
```

Date: self.__nr = ["5", "7", "7", "1", "g", "5"] bs = 10
nr.convert_by_substitution(15)

Rezultat: self.__nr = ['B', "6", "0", "4", "A"]

Precondiții: numărul stocat sub formă de obiect nr. se aplică subalgoritmul
baza nouă > baza veche

Postcondiții: numărul stocat în nr.. va fi cel în noua bază

subalgoritm convert_by_division(self, new_base) - conversie prin împărțire
    dacă (new_base < self.__base)
        new_ns = []
        cât timp self.__ns != []: execută
            rez = []
            r = 0
            [rez, r] = self.div_by_digit(self.digit(new_base))
            self.__ns = str(rez)
            new_ns.append(r)
        self.__base = new_base
        new_ns.reverse()
        self.__ns = (new_ns) - getează la stringuri

Date: self.__ns = ["A", "B"] bas = 16
    ns.convert_by_division(10)

Rezultat: self.__ns = ["1", "7", "1"]

Precondiții: nr. stocat ca obiect number asupra căruia aplicăm subalgoritmul și baza nouă < baza veche

Postcondiții: în cadrul obiectului nr se va afla numărul în noua bază

subalgoritm conversion_using_intermediary_base(self, new_base) - conversia folosind o bază intermediară

```
exp = 1
rez = 0
self.__nr.reverse()
for elem in self.__nr:
    rez += exp * get_digit(elem)
    exp *= self.__base
new_nr = []
cât timp rez != 0 execută
    rest = rez % new_base
    rez //= new_base
    new_nr.append(str_digit(rest))
new_nr.reverse()
self.__nr = new_nr
self.__base = new_base
```

Date: self.__nr = ["2", ",", "0", ",", "1"] bs = 3
nr.conversion_using_intermediary_base(7)

Rezultat: self.__nr = ['1', '2', ',', '1']

Precondiții: nr. stocat sub formă de obiect number asupra căreia se aplică funcții built-in

Postcondiții: în nr. va fi stocat numărul în noua bază

Date: self.__nr = ['1', '0', '1', '0', '0', '0', '1', '0', '1', '1', '1', '1', '0', '0', '0'] bs = 2
nr.fast_conversions(16)

Rezultat: self.__nr = ['A', '2', 'F', '0']

Precondiții: nr. tip obiect number. conversia se realizează cu funcții built-in din baza 2 în 4, 8, 16 și din 4, 8, 16 în baza

Postcondiții: nr. se va afla în self.__nr.

```
def algoritm_fast_conversie(self, new_baza) - conversii rapide
    daca self.__baza == 2
        ord = {lg(n) / lg(new_baza)}
        self.__nr.reverse()
        ln = len(self.__nr)
        self.__nr += {'0'} * 4
        index = 0
        rez = []
        while index < ln:
            daca ord = 2 atunci
                rez.append(set_digit({index} + 2*{index+1}))
                index += 2
            daca ord = 3 atunci
                rez.append(set_digit({index} + 2*{index+1} + 4*{index+2}))
                index += 3
            daca ord = 4 atunci
                rez.append(set_digit({index} + 2*{index+1} + 4*{index+2} + 8*{index+3}))
                index += 4
        rez.elimina_0()
        rez.reverse
        self.__nr = rez
        self.__baza = new_baza
    daca new_baza == 2 atunci
        ord = {lg(n) / lg(new_baza)}
        cpy = self.__baza
        for elem in self.__nr:
            partial_rez = {}
            cat timp elem != 0 executa
                elem = get_digit(elem)
                partial_rez.append(elem%2)
                elem //= 2
            calculam lungimea
            partial_rez.reverse()
            rez += partial_rez
        rez.elimina_0()
        self.__nr = rez
        self.__baza = new_baza
```

Set de test.

(1)
1
+(-)(*)(/)
A
16
10
2
10
y

10(10) + 2(10) = 12(10)
(10(10) - 2(10) = 8(10))
(10(10) · 2(10) = 20(10))
(10(10) : 2(10) = 5(10) rest 2(10))

(2) 2

1	2	3	4
A	B	2101	A2F0
16	16	3	16
10	10	I	2
y	y	y	y
10(10)	11(10)	121(7)	1010001011110000(2)

Cerințe minime: am atașat fișierul main.py acesta poate fi rulat direct în cmd prin comanda „python 3 main.py". Dacă se pune în problem se poate observa că aplicația are funcții de testare ce asigură corectitudinea fiecărei funcții