Timeo Delcroix_Salomé Ruiz_Cassandre Chanay

Projet Python

Projet 1 : les bases

15 novembre 2024

Introduction

Ce projet consiste à écrire un code permettant de transformer un nombre en base n en dans une autre base m. Le programme permet de passer entre les bases 2, 4 et 16.

Nos différentes fonctions principales

bin_dec_hex_to_bin_dec_hex

check_compatibility_number_base

transform

Première fonction principale

main.py > ...

```python
from utils import *  # Assurez-vous que utils.py contient les bonnes fonctions

def bin_dec_hex__to__bin_dec_hex(init_number, init_base, target_base):

    '''
      Cette fonction prend en compte le nombre d'entrée,
      la base de départ et la base souhaitée. Elle va donc lancée la
      fontion ci-dessous correspondante.
    '''
```

```python
if init_base == 2 and target_base == 2:
    return bin_to_bin(init_number)

if init_base == 2 and target_base == 10:
    return bin_to_dec(init_number)

if init_base == 2 and target_base == 16:
    return bin_to_hex(init_number)

if init_base == 10 and target_base == 2:
    return dec_to_bin(init_number)

if init_base == 10 and target_base == 10:
    return dec_to_dec(init_number)

if init_base == 10 and target_base == 16:
    return dec_to_hex(init_number)

if init_base == 16 and target_base == 2:
    return hex_to_bin(init_number)

if init_base == 16 and target_base == 10:
    return hex_to_dec(init_number)

if init_base == 16 and target_base == 16:
    return hex_to_hex(init_number)

return None
```

Voici toutes les possibilités suivant les bases initiales et finales. La 1ère fonction prend en compte tous ces choix afin d'exécuter le programme qui correspond.

Deuxième fonction principale

```python
def transform():

    '''
    Cette fonction demande les valeurs de base, de nombre et de base cible à l'utilisateur
    puis effectue la conversion et affiche le résultat.
    '''
    init_number = ask_for_init_number()
    init_base = ask_for_init_base()
    target_base = ask_for_target_base()
    # check_compatibility_number_base

    final_number = bin_dec_hex__to__bin_dec_hex(init_number, init_base, target_base)
    print (f"Le nombre converti est : {final_number}")

if __name__ == "__main__":
    transform()
```

Quelques exemples de cette fonction:

```
Entrez le nombre : 26
Entrez la base de départ : 10
Entrez la base souhaitée : 2
Le nombre converti est : 11010
PS C:\Users\Cassandre Chanay\C
3.12.exe" "c:/Users/Cassandre
Entrez le nombre : 1001
Entrez la base de départ : 2
Entrez la base souhaitée : 10
Le nombre converti est : 9
PS C:\Users\Cassandre Chanay\C
3.12.exe" "c:/Users/Cassandre
Entrez le nombre : 12A
Entrez la base de départ : 16
Entrez la base souhaitée : 10
Le nombre converti est : 298
PS C:\Users\Cassandre Chanay\C
```

Troisième fonction principale

```python
def check_compatibility_number_base(init_number, init_base):
    while True:
        if init_base == "2":
            if all(char in bin_number_valid_chars for char in init_number):
                return init_number
            else:
                print(error_in_init_base)
                init_number = ask_for_init_number()
        elif init_base == "10":
            if all(char in dec_number_valid_chars for char in init_number):
                return init_number
            else:
                print(error_in_init_base)
                init_number = ask_for_init_number()
        elif init_base == "16":
            if all(char in hex_number_valid_chars for char in init_number):
                return init_number
            else:
                print(error_in_init_base)
                init_number = ask_for_init_number()
```

Voici un exemple du fonctionnemnt de cette fonction :

```
Entrez le nombre : 54
Entrez la base de départ : 2
La base d'entrée ne correspond
pas au chiffre de départ.
Veuillez recommencer.
Entrez le nombre : 10
Entrez la base souhaitée : 16
Le nombre converti est : 2
```

On remarque que la vérification a lieu après l'entrée de la base initiale. Si la base ne correspond pas au nombre de départ, le programme redemande le nombre initiale dans la base de départ choisie. On peut ensuite reprendre l'éxécution du programme normalement.

Nos fonctions de conversions numériques

bin_to_bin

hex_to_hex

dec_to_dec

I- D’une base vers la même

bin_to_hex

hex_to_bin

III- D’un extrême à l’autre

bin_to_dec

hex_to_dec

dec_to_bin

dec_to_hex

II- D’une base à une autre

I- D'une base vers la même

```python
def bin_to_bin(init_number):
    return init_number
```

bin_to_bin

- Si le nombre entré est en binaire, la base de départ binaire et la base finale binaire, bin_to_bin renvoie le nombre entré puisqu'il n'y a aucun changement de base à effectuer.

```python
def dec_to_dec(init_number):
    return init_number
```

dec_to_dec

- Si le nombre entré est en décimal, la base de départ décimale et la base finale décimale, dec_to_dec renvoie le nombre entré puisqu'il n'y a aucun changement de base à effectuer.

```python
def hex_to_hex(init_number):
    return init_number
```

hex_to_hex

- Si le nombre entré est en hexadécimal, la base de départ hexadécimale et la base finale hexadécimale, dec_to_dec renvoie le nombre entré puisqu'il n'y a aucun changement de base à effectuer.

Ces fonctions renvoient l'argument 'init_number' sans aucune modification.

II- D’une base à une autre

dec_to_bin :

```
def dec_to_bin(init_number):
    init_number = int(init_number)              # pour convertir init_number en entier.
    if init_number == 0:                        # car 0 en base 10 est aussi 0 en base 2.
        return "0"
    restes = ""                                 # chaîne vide qui accumule les chiffres binaires formant le résultat final.
    while init_number > 0:                      # à chaque itération :
        restes = str(init_number % 2) + restes  # on calcule le reste de la division de init_number par 2,
#                                                 qui donne le chiffre binaire le plus à droite.
#                                                 Le reste est converti en chaîne (str) et ajouté au début de la chaîne
#                                                 restes (pour avoir les chiffres dans le bon ordre).
        init_number //= 2                       # init_number est ensuite mis à jour en le divisant par 2.
    return restes                               # la fonction retourne la chaîne restes, qui contient le nombre en base 2.
```

II- D’une base à une autre

dec_to_hex :

```
def dec_to_hex(init_number):
    init_number = int(init_number)          # pour convertir init_number en entier.
    if init_number == 0:                    # car 0 en base 10 est aussi 0 en base 16.
        return "0"
    hex_digits = "0123456789ABCDEF"         # hex_digits contient des caractères hexadécimaux (valeurs de 0 à 15),
#                                           # => permet la conversion des restes en leurs représentations hexadécimales.
    restes = ""                             # chaîne vide qui accumule les chiffres hexadécimaux formant le résultat final.
    quotient = init_number                  # quotient est initialisé avec la valeur de init_number car il sera modifié dans la boucle.
    while quotient > 0:                     # à chaque itération :
        reste = quotient % 16               # on calcule le reste de la division de quotient par 16
        restes = hex_digits[reste] + restes # reste: utilisé comme indice pour obtenir le caractère héxadécimal correspondant dans hex_digits
#                                             et ce caractère est ajouté au début de la chaine restes (pour avoir les chiffres dans le bon ordre)
        quotient //= 16                     # pour mettre le quotient à jour en le divisant par 16 pour préparer l'itération suivante.
    return restes                           # la fonction retourne la chaîne restes, qui contient le nombre en base 16.
```

II- D’une base à une autre

bin_to_dec :

```
bin_to_dec(init_number):
result = 0
exposant = len(init_number) - 1    # Calcule l'exposant initial, correspondant à la position du bit le plus à gauche (plus significatif).
for char in init_number:
    if char == '1':                # Vérifie si le bit actuel est '1'.
        result += 2 ** exposant    # Si oui, ajoute la valeur de 2 élevé à `exposant` à `result`
    exposant -= 1                  # Réduit l'exposant de 1 pour passer au bit suivant, vers la droite.
return result                      # Renvoie la valeur finale de "init_number"
```

II- D’une base à une autre

hex_to_dec :

```
def hex_to_dec(init_number):
    decimal_value = 0                                                    # Initialiser la valeur décimale à 0
    hex_liste = {
        '0': 0, '1': 1, '2': 2, '3': 3, '4': 4,
        '5': 5, '6': 6, '7': 7, '8': 8, '9': 9,
        'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15
    }                                                                    # Liste qui associe chaque caractère hexadécimal
#                                                                          à sa valeur en base 10

    for caracter in init_number:                                         # Parcourir chaque caractère du nombre initial en base 16
        if caracter not in hex_liste:
            print("Erreur : Le nombre entré n'est pas valide en base 16.")  # Vérifier si le caractère est valide en vérifiant
#                                                                          qu'il est bien présent dans la liste hex_liste

        decimal_value = decimal_value * 16 + hex_liste[caracter]         # Mettre à jour la valeur décimale initialisée à 0
#                                                                          en multipliant par 16 et en ajoutant la valeur en base 10
#                                                                          du nombre initial trouvé la liste hex_liste

    return decimal_value                                                 # Retourner la valeur décimale actualisée
```

III- D’un extrême à l’autre

```
def hex_to_bin(init_number):
    init_number = hex_to_dec(init_number)
    return dec_to_bin(init_number)
```

Pour passer un nombre en base héxadécimale à un nombre en base binaire, la fonction hex_to_bin convertit dans un premier temps le nombre en base décimale et dans un second temps le nombre en base binaire.

Pour passer un nombre en base binaire à un nombre en base héxadécimale, la fonction bin_to_hex convertit dans un premier temps le nombre en base décimale et dans un second temps le nombre en base héxadécimale.

```
def bin_to_hex(init_number):
    init_number = bin_to_dec(init_number)
    return dec_to_hex(init_number)
```

La réutilisation des fonctions de calculs précédemment écrites nous a permis un gain de temps, une minimisation des complexités et l’affranchissement de la création de nouvelles fonctions de calculs.

Conclusion

Nous sommes passé par différentes fonctions et sous fonctions afin de perfectionner au maximum notre programme, ainsi que pour éviter tous les potentiels problèmes surtout par rapport aux bases et aux nombres initiaux.

Merci pour votre attention