Part A

Assume that the MPIN is 4-digits. Write a program that suggests if the MPIN is a commonly used one.

In [1]:
```
import itertools
from itertools import permutations
```

Logic used

- Created functions "is_common()" to catch common patterns of mpin.
- Common pattern include- palindromes, repeted digitis, liner increase in digits, or easy to remember patterns
- Function 'partA' tests the input mpins as either commonly used mpins or not commonly used pins.

In [2]:
```
def is_common(pin: str) -> bool:
    if len(pin) == 4:
        common = [ '0123', '1234', '4567', '7890', '5678']
        return True if pin in common else False

    elif len(pin) == 6:
        common = [
                "123456",
                "000000",
                "111111",
                "121212",
                "654321",
                "112233",
                "999999",
                "123123",
                "159753",
                "111222"
        ]

def is_palin(pin: str) -> bool:
    return pin[::-1] == pin

def is_repeated(pin: str) -> bool:
    if int(pin[:2]) == int(pin[2:]) - 11 or int(pin[:2]) - 11 == int(pin[2:]): r
    return pin[:2] == pin[2:]
```

In [3]:
```
def partA(pin: str) -> bool:
    if is_common(pin) or is_palin(pin) or is_repeated(pin): print("It is a commo
    else: print("It is not a commonly used pin")
    return True
```

Part B

Enhance the above to take user's demographics as input and provides an output

- a. Strength: WEAK or STRONG

Logic used

- Created a hashmap (dictionary within dictionary) called months and assigned two keys-
  1. Numeric representation of months
  2. Number of days of the months
- Created function called-
  1. convert_month used to convert the input from Jan -> 01, Feb -> 02, and so on
  2. returnArray used to return array with demographs if found any in the permutations or combinations, for the question in Part B, there is no need to return demographs hence array is not called.
  3. generate_all_pattern - Returns array/list, used to generate all possible set of patterns without repetation to check if mpin is formed with the current demograph
  4. Function 'partB' tests the input mpins on conditions- commonly used mpins, demographs and returns WEAK or STRONG

In [4]:
```
months = {
    "Jan": {"num": "01", "days": 31},
    "Feb": {"num": "02", "days": 28},
    "Mar": {"num": "03", "days": 31},
    "Apr": {"num": "04", "days": 30},
    "May": {"num": "05", "days": 31},
    "Jun": {"num": "06", "days": 30},
    "Jul": {"num": "07", "days": 31},
    "Aug": {"num": "08", "days": 31},
    "Sep": {"num": "09", "days": 30},
    "Oct": {"num": "10", "days": 31},
    "Nov": {"num": "11", "days": 30},
    "Dec": {"num": "12", "days": 31}
}
```

In [5]:
```
def convert_month(month: str) -> str:
    month = month.strip().title()
    if month not in months: return False
    return months[month]['num']
```

In [6]:
```
def returnArray(pin: str, dob_self: list, dob_spouse: list, anniversary: list) -
    day_self, month_self, year_self = dob_self[0], convert_month(dob_self[1]), d
    day_spouse, month_spouse, year_spouse = dob_spouse[0], convert_month(dob_spo
    day_anni, month_anni, year_anni = anniversary[0], convert_month(anniversary[

    arr = []

    self, spouse, anni, all_combined = [], [], [], []
    self.append(day_self), self.append(month_self), self.append(year_self)
    spouse.append(day_spouse), spouse.append(month_spouse), spouse.append(year_s
    anni.append(day_anni), anni.append(month_anni), anni.append(year_anni)
    all_combined = [self, spouse, anni]

    #single check
```

```
    self_pattern = generate_all_pattern(self, len(pin))
    spouse_pattern = generate_all_pattern(spouse, len(pin))
    anni_pattern = generate_all_pattern(anni, len(pin))


    #Demographs
    if is_common(pin) or is_palin(pin) or is_repeated(pin):
        arr.append("COMMONLY_USED")
    if pin in self_pattern:
        arr.append("DEMOGRAPHIC_DOB_SELF")
    if pin in spouse_pattern:
        arr.append("DEMOGRAPHIC_DOB_SPOUSE")
    if pin in anni_pattern:
        arr.append("DEMOGRAPHIC_ANNIVERSARY")

    arr = set(arr)
    return list(arr)

def generate_all_pattern(self: list, size: int) -> list:
    if size == 4:
        self[2] = self[2][-2:]
        pattern = []
        for i in permutations(self, 2):
            if len(''.join(i)) == 4:
                pattern.append(i)
        pattern_string = [''.join(p) for p in pattern]
        return list(set(pattern_string))

    if size == 6:
        l1 = self
        strip_l1_year = l1[2][-2:]
        new_l1 = l1 + [strip_l1_year]
        pattern = []

        for i in permutations(new_l1, 2):
            if len(''.join(i)) == 6:
                pattern.append(i)
        for i in permutations(new_l1, 3):
            if len(''.join(i)) == 6:
                pattern.append(i)

        pattern_string = list(set([''.join(p) for p in pattern]))
        return list(set(pattern_string))
```

4 Digit Mpin without demographics

In [7]:

```
def partB(pin, dob_self, dob_spouse, anniversary):
    dob_self = dob_self.replace(" ", "").split("-")
    dob_spouse = dob_spouse.replace(" ", "").split("-")
    anniversary = anniversary.replace(" ", "").replace(" ", "").split("-")

    test_for_4 = returnArray(pin, dob_self, dob_spouse, anniversary)

    if len(test_for_4) !=0: print("Strength: WEAK")
    else: print("Strength: STRONG")
```

Part C

Enhance the above to provide the following outputs

- a. Strength: WEAK or STRONG
- b. If weak then the reason why was it considered weak: It should give from the following the reasons as an array.
- Array should be empty if Strength is STRONG and non-empty if WEAK
- COMMONLY_USED
- DEMOGRAPHIC_DOB_SELF
- DEMOGRAPHIC_DOB_SPOUSE
- DEMOGRAPHIC_ANNIVERSARY

Logic used

- Created function-
  1. find_match_in_two - Returns True if the pattern is matched based on permutations of two demographs
  2. find_match_in_three - Returns True if the pattern is matched based on permutations of three demographs
  3. Function 'partC' tests the input mpins on conditions- commonly used mpins, demographs and returns WEAK or STRONG along with the name of the combinations of demographs used to form mpins

In [8]:
```
def return_array(pin: str, dob_self: list, dob_spouse: list, anniversary: list)
    day_self, month_self, year_self = dob_self[0], convert_month(dob_self[1]), d
    day_spouse, month_spouse, year_spouse = dob_spouse[0], convert_month(dob_spo
    day_anni, month_anni, year_anni = anniversary[0], convert_month(anniversary[

    arr = []

    self, spouse, anni = [], [], []
    self.append(day_self), self.append(month_self), self.append(year_self)
    spouse.append(day_spouse), spouse.append(month_spouse), spouse.append(year_s
    anni.append(day_anni), anni.append(month_anni), anni.append(year_anni)

    #single check
    self_pattern = generate_all_pattern(self, len(pin))
    spouse_pattern = generate_all_pattern(spouse, len(pin))
    anni_pattern = generate_all_pattern(anni, len(pin))

    #double Check
    self_spouse_pattern_double = find_match_in_two(self, spouse, pin, len(pin))
    spouse_anni_pattern_double = find_match_in_two(spouse, anni, pin, len(pin))
    anni_self_pattern_double = find_match_in_two(anni, self, pin, len(pin))

    #triple check
    self_pattern_triple = find_match_in_three(self, spouse, anni, pin, len(pin))

    #Demographs

    if is_common(pin) or is_palin(pin) or is_repeated(pin):
        arr.append("COMMONLY_USED")

    if pin in self_pattern: #.....................................................
```

```
            arr.append("DEMOGRAPHIC_DOB_SELF")
        if pin in spouse_pattern:
            arr.append("DEMOGRAPHIC_DOB_SPOUSE")
        if pin in anni_pattern:
            arr.append("DEMOGRAPHIC_ANNIVERSARY")

        if self_spouse_pattern_double: #.....................................
            #print("match found double 1")
            arr.append("DEMOGRAPHIC_DOB_SELF")
            arr.append("DEMOGRAPHIC_DOB_SPOUSE")
        if spouse_anni_pattern_double:
            #print("match found double 2")
            arr.append("DEMOGRAPHIC_DOB_SPOUSE")
            arr.append("DEMOGRAPHIC_ANNIVERSARY")
        if anni_self_pattern_double:
            #print("match found double 3")
            arr.append("DEMOGRAPHIC_ANNIVERSARY")
            arr.append("DEMOGRAPHIC_DOB_SELF")

        if self_pattern_triple: #.............................................
            #print("match found triple")
            arr.append("DEMOGRAPHIC_DOB_SELF")
            arr.append("DEMOGRAPHIC_DOB_SPOUSE")
            arr.append("DEMOGRAPHIC_ANNIVERSARY")

        arr = set(arr)
        return list(arr)

    def find_match_in_two(l1: list, l2: list, pin: str, size: int) -> bool:
        if size == 6:
            strip_l1_year = l1[2][-2:]
            strip_l2_year = l2[2][-2:]
            new_l1 = l1 + [strip_l1_year]
            new_l2 = l2 + [strip_l2_year]

            combined = new_l1 + new_l2
            combination = []

            for p in permutations(combined, 2):
                joined = ''.join(p)
                if len(joined) == 6:
                    combination.append(joined)
            for p in itertools.permutations(combined, 3):
                joined = ''.join(p)
                if len(joined) == 6:
                    combination.append(joined)


            product = [''.join(c) for c in combination if len(''.join(c)) == 6]
            product = list(set(product))

            if pin in set(combination): return True
            return False

        if size == 4:
            l1[2] = l1[2][-2:]
            l2[2] = l2[2][-2:]

            combination1 = list(itertools.product(l1, l2))
            combination2 = list(itertools.product(l2, l1))
```

```
        combination = combination1 + combination2
        product = [''.join(p) for p in combination]
        product = list(set(product))

        if pin in product: return True
        return False

def find_match_in_three(l1: list, l2: list, l3: list, pin: str, size: int) -> bo
    if size == 6:
        strip_l1_year = l1[2][-2:]
        strip_l2_year = l2[2][-2:]
        strip_l3_year = l3[2][-2:]
        new_l1 = l1 + [strip_l1_year]
        new_l2 = l2 + [strip_l2_year]
        new_l3 = l3 + [strip_l3_year]

        combination1 = list(itertools.product(new_l1, new_l2, new_l3))
        combination2 = list(itertools.product(new_l1, new_l3, new_l2))
        combination3 = list(itertools.product(new_l2, new_l1, new_l3))
        combination4 = list(itertools.product(new_l2, new_l3, new_l1))
        combination5 = list(itertools.product(new_l3, new_l1, new_l2))
        combination6 = list(itertools.product(new_l3, new_l2, new_l1))

        combination = combination1 + combination2 + combination3 + combination4
        product = [''.join(c) for c in combination if len(''.join(c)) == 6]
        list(set(product))
        if pin in product: return True
        return False
    return False
```

Part C-4 digit with demographics

In [9]:

```
def partC(pin: str, dob_self: str, dob_spouse: str, anniversary: str) -> bool:
    dob_self = dob_self.replace(" ", "").split("-")
    dob_spouse = dob_spouse.replace(" ", "").split("-")
    anniversary = anniversary.replace(" ", "").split("-")

    test_for_4 = return_array(pin, dob_self, dob_spouse, anniversary)

    if len(test_for_4) != 0:
        print("Strength: Weak", test_for_4)
    else:
        print("Strength: Strong", test_for_4)

    return True
```

Part D

Above with a 6-digit PIN

Logic used

- updated earlier functions such that the input size for mpin become 6 digit.
- Updated logical process in find_match_in_three and find_match_in_two such that permutations are formed using parts-

1. 2 part -> (day, year(last 4 digit)) , (month, year(last 4 digit), etc)
2. 3 part -> (day, month, year(last 2 digit))

In [10]:
```
def partD(pin: str, dob_self: str, dob_spouse: str, anniversary: str) -> bool:
    dob_self = dob_self.replace(" ", "").split("-")
    dob_spouse = dob_spouse.replace(" ", "").split("-")
    anniversary = anniversary.replace(" ", "").split("-")

    test_for_6 = return_array(pin, dob_self, dob_spouse, anniversary)

    if len(test_for_6) != 0:
        print("Strength: Weak", test_for_6)
    else:
        print("Strength: Strong", test_for_6 )

    return True
```

Test Cases

Part A - 20 Cases

In [11]:
```
mpin = ["0000", "1234", "4321", "1111", "1212", "0007", "7890", "2468", "1357",
for i in mpin:
    partA(i)
```

```
It is a commonly used pin
It is a commonly used pin
It is not a commonly used pin
It is a commonly used pin
It is a commonly used pin
It is not a commonly used pin
It is a commonly used pin
It is not a commonly used pin
It is not a commonly used pin
It is a commonly used pin
It is not a commonly used pin
It is not a commonly used pin
It is a commonly used pin
It is a commonly used pin
It is a commonly used pin
It is a commonly used pin
It is not a commonly used pin
It is not a commonly used pin
It is a commonly used pin
It is not a commonly used pin
```

Part B- 20 Cases

In [12]:
```
mpin = ["0000", "1234", "4321", "1111", "1212", "0007", "7890", "2468", "1357",
        "9876", "1023", "9999", "2110", "7788", "1221", "2308", "3456", "8080",

dob_self = [
    "01-Jan-2000", "04-Apr-1990", "07-Jul-1988", "10-Oct-1975", "13-Jan-1960",
    "16-Apr-2003", "19-Jul-2011", "22-Oct-1980", "25-Jan-1970", "28-Apr-1965",
    "01-Jul-2001", "04-Oct-1999", "07-Jan-2005", "10-Apr-2010", "13-Jul-2013",
```

```
    "16-Oct-2016", "19-Jan-2019", "22-Apr-2022", "25-Jul-2025", "28-Oct-2028"
]

dob_spouse = [
    "02-Feb-2001", "05-May-1992", "08-Aug-1989", "11-Nov-1976", "14-Feb-1961",
    "17-May-2004", "20-Aug-2012", "23-Nov-1981", "26-Feb-1971", "29-May-1966",
    "02-Aug-2002", "05-Nov-2000", "08-Feb-2006", "11-May-2011", "14-Aug-2014",
    "17-Nov-2017", "20-Feb-2020", "23-May-2023", "26-Aug-2026", "29-Nov-2029"
]

anniversary = [
    "03-Mar-2010", "06-Jun-2015", "09-Sep-2000", "12-Dec-2005", "15-Mar-1999",
    "18-Jun-2020", "21-Sep-2022", "24-Dec-2010", "27-Mar-1995", "30-Jun-1988",
    "03-Sep-2021", "06-Dec-2023", "09-Mar-2010", "12-Jun-2012", "15-Sep-2015",
    "18-Dec-2018", "21-Mar-2021", "24-Jun-2024", "27-Sep-2027", "30-Dec-2030"
]

for i in range(len(mpin)):
    partB(mpin[i], dob_self[i], dob_spouse[i], anniversary[i])
```

```
Strength: WEAK
Strength: WEAK
Strength: STRONG
Strength: WEAK
Strength: WEAK
Strength: STRONG
Strength: WEAK
Strength: STRONG
Strength: STRONG
Strength: WEAK
Strength: STRONG
Strength: STRONG
Strength: WEAK
Strength: WEAK
Strength: WEAK
Strength: WEAK
Strength: STRONG
Strength: STRONG
Strength: WEAK
Strength: STRONG
```

Part C- 20 Cases

In [13]:
```
for i in range(len(mpin)):
    partC(mpin[i], dob_self[i], dob_spouse[i], anniversary[i])
```

```
Strength: Weak ['COMMONLY_USED']
Strength: Weak ['COMMONLY_USED']
Strength: Strong []
Strength: Weak ['COMMONLY_USED', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['COMMONLY_USED']
Strength: Strong []
Strength: Weak ['COMMONLY_USED']
Strength: Strong []
Strength: Strong []
Strength: Weak ['COMMONLY_USED']
Strength: Strong []
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY']
Strength: Weak ['COMMONLY_USED']
Strength: Weak ['COMMONLY_USED']
Strength: Weak ['COMMONLY_USED']
Strength: Weak ['COMMONLY_USED']
Strength: Strong []
Strength: Strong []
Strength: Weak ['COMMONLY_USED']
Strength: Strong []
```

Part D- 20 Cases

In [14]:
```
mpin = ["010200", "040492", "078988", "101175", "136061", "160604", "190811", "2
        "010802", "041099", "070105", "101210", "131313", "161618", "192021", "2

for i in range(len(mpin)):
    partD(mpin[i], dob_self[i], dob_spouse[i], anniversary[i])
```

```
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_DOB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY']
Strength: Strong []
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY', 'DEMOGRAPHIC_D
OB_SPOUSE']
Strength: Strong []
Strength: Weak ['DEMOGRAPHIC_DOB_SELF', 'DEMOGRAPHIC_ANNIVERSARY']
```