

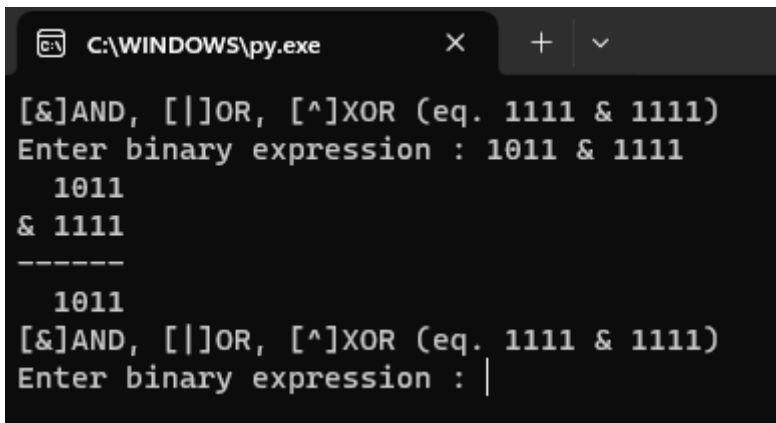
Python

Snippets/Scripts

- [Bitwise OR XOR AND Calculator](#)
- [Decimal to Binary Converter](#)
- [Decimal to Hex Converter](#)
- [2s Compliment Calculator](#)
- [Inheritance](#)
- [Create SQLite DB from Wikipedia](#)
- [Pandas](#)
 - [Set max row for dataframe output](#)
 - [Combine df with concat](#)
 - [Create histogram from df](#)

Bitwise OR XOR AND Calculator

Example:



```
C:\WINDOWS\py.exe X + v
[&]AND, [|]OR, [^]XOR (eq. 1111 & 1111)
Enter binary expression : 1011 & 1111
  1011
& 1111
-----
  1011
[&]AND, [|]OR, [^]XOR (eq. 1111 & 1111)
Enter binary expression : |
```

Code:

```
def valid_expression(expression: str) -> bool:
    """ Returns True if string is valid binary expression. False otherwise.

    - Operand is valid binary digits
    - Operator is either '!', '&', or '|'
    - Length of operands is more than 2 digits
    - Both operands are same length
    - Expression is in the format of "operand operator operand", seperated by blanks"""
    if len(expression.split()) == 3:
        first, operator, second = expression.split()
    else:
        print("Format must be "operand operator operand", separated by blanks")
        return False
    for i in range(len(first)):
        if not first[i] in '10' or not second[i] in '10' : # If digit is not 1 or 0
            print('Operand does not consist of binary digits')
            return False
```

```

if not operator in '&^|':
    print(f'{operator} is invalid. Must be !, & or ^')
    return False
elif len(first) < 2 or len(second) < 2:
    print('Length of the operands must be at least 2 digits')
    return False
elif len(first) != len(second):
    print('The operands are of different length')
    return False
return True

def calc_binary_expression(firstBinary: str, operator: str, secondBinary: str) -> str:
    """ Returns the result (str) of the binary expression. """
    result = ""
    for i in range(0, len(firstBinary)):
        z = 0
        if operator == "&":
            z = 1 if int(firstBinary[i]) and int(secondBinary[i]) else 0
        elif operator == "|":
            z = 1 if int(firstBinary[i]) or int(secondBinary[i]) else 0
        elif operator == "^":
            z = 1 if int(firstBinary[i]) != int(secondBinary[i]) else 0
        result = str(result) + str(z)
    return result

def display_bin_calculation(firstBinary: str, operator: str, secondBinary: str) -> None:
    """ Print the result of the binary and operator from parameter """
    print(f"{'':<2}{firstBinary}")
    print(f"{'':<2}{operator} {secondBinary}")
    print(f"{'':<2}{calc_binary_expression(firstBinary, operator, secondBinary)}")

def main() -> None:
    """ This program reads in a binary expression as a string and evaluates the result. """
    while True:
        userInput = input("[&]AND, [|]OR, [^]XOR (eq. 1111 & 1111) \nEnter binary expression : ")

        # Without any input, break the loop
        if len(userInput) == 0:
            print("End of program.")

```

```
break
```

```
if valid_expression(userInput):
```

```
    firstBinary, operator, secondBinary = userInput.split()
```

```
    display_bin_calculation(firstBinary, operator, secondBinary)
```

```
if __name__ == "__main__":
```

```
    main()
```

Decimal to Binary Converter

Example:

```
C:\WINDOWS\py.exe
Enter decimal value : 129

Division by 2    Quotient    Remainder    Bit #
129/2            64           1            0
64/2             32           0            1
32/2             16           0            2
16/2             8            0            3
8/2              4            0            4
4/2              2            0            5
2/2              1            0            6
1/2              0            1            7

129 (base 10) = 10000001 (base 2) [bitlength = 8]

Enter decimal value :
```

Code:

```
def decToBinLongDivision(decimal: int) -> None:
    numtodivide = decimal
    quotient = 0
    remainder = 0
    bitlength = 0
    strBinary = ""

    if decimal == 0:
        strBinary = '0'

    print('Division by 2\tQuotient\tRemainder\tBit #')
    while True:
        if numtodivide <= 0:
            break
```

```

quotient = int(numtodivide / 2)
remainder = numtodivide % 2
print(f'{str(numtodivide)}+ "/2":<8}\t{quotient:<8}\t{remainder:<8}\t{bitlenght:<8}')
numtodivide = quotient
bitlenght += 1
strBinary += str(remainder)
print(f'\n{decimal} (base 10) = {strBinary[::-1]} (base 2) [bitlength = {bitlenght}]\n')

```

```
def main() -> None:
```

```
    """ This program reads in a binary expression as a string and evaluates the result. """
```

```
    while True:
```

```
        userInput = input("Enter decimal value : ")
```

```
        # Without any input, break the loop
```

```
        if len(userInput) == 0:
```

```
            print("End of program.")
```

```
            break
```

```
        print('\n')
```

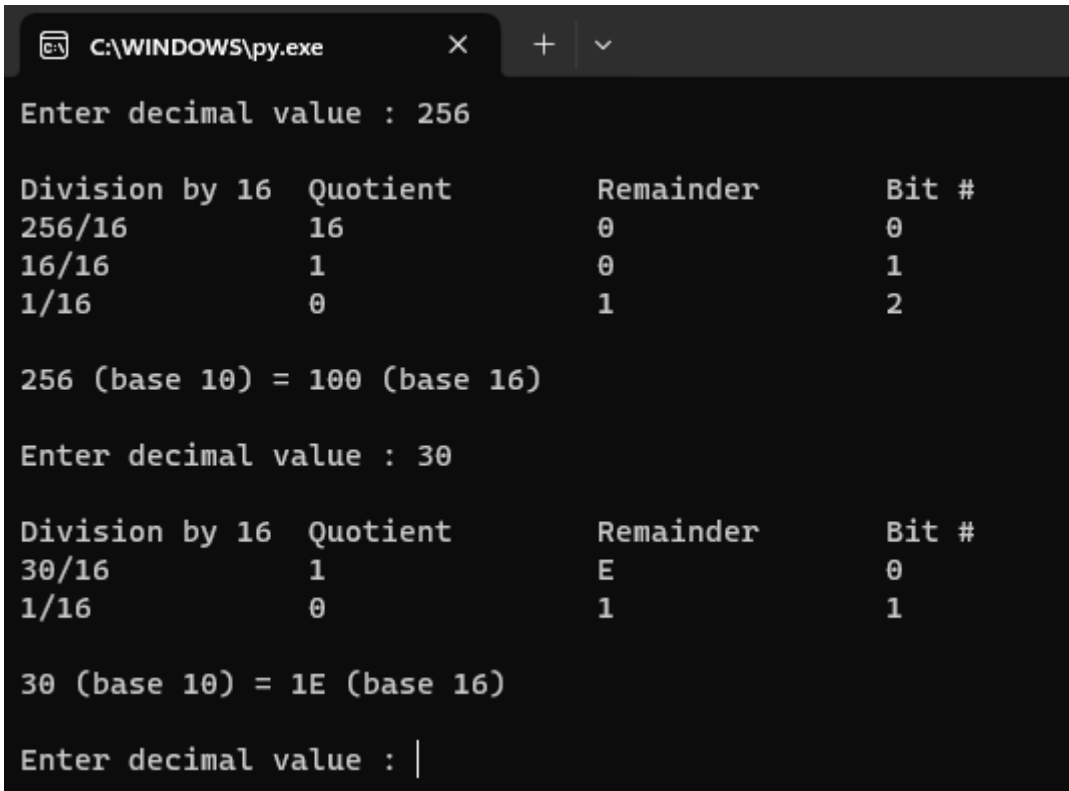
```
        decToBinLongDivision(int(userInput))
```

```
if __name__ == "__main__":
```

```
    main()
```

Decimal to Hex Converter

Example:



```
C:\WINDOWS\py.exe
Enter decimal value : 256

Division by 16  Quotient      Remainder    Bit #
256/16         16          0           0
16/16          1           0           1
1/16           0           1           2

256 (base 10) = 100 (base 16)

Enter decimal value : 30

Division by 16  Quotient      Remainder    Bit #
30/16           1           E           0
1/16            0           1           1

30 (base 10) = 1E (base 16)

Enter decimal value : |
```

Code:

```
def decToHexLongDivision(decimal: int) -> str:
    numtodivide = decimal
    quotient = 0
    remainder = 0
    bitlength = 0
    strBinary = ""

    print("\nDivision by 16\tQuotient\tRemainder\tBit #')
    while True:
        if numtodivide <= 0:
            break
        quotient = int(numtodivide / 16)
```

```
remainder = hex(numtodivide % 16).split('x')[-1].upper()
print(f'{str(numtodivide)+"/16":<8}\t{quotient:<8}\t{remainder:<8}\t{bitlenght:<8}')
```

numtodivide = quotient

bitlenght += 1

strBinary += str(remainder)

```
print(f'\n{decimal} (base 10) = {strBinary[::-1]} (base 16)\n')
```

```
def main() -> None:
```

```
    """ This program reads in a binary expression as a string and evaluates the result. """
```

```
    while True:
```

```
        userInput = input("Enter decimal value : ")
```

```
        # Without any input, break the loop
```

```
        if len(userInput) == 0:
```

```
            print("End of program.")
```

```
            break
```

```
        decToHexLongDivision(int(userInput))
```

```
if __name__ == "__main__":
```

```
    main()
```


2s Compliment Calculator

Example:

```
C:\WINDOWS\py.exe x + v
Enter decimal value: -127
Enter Total Bit Length (including sign bit): 8

-----
1. Convert to Binary
If negative decimal, change to positive decimal into 7-bit binary first
Decimal:          -127
+ve Decimal:       127
Binary:            01111111
removesignbit:    1111111

-----
2. Inverting the bit
invert binary:    0000000

-----
3. Add 1 complement
+1 Binary:       0000001

-----
4. Apply negative sign to the MSB
2scomplement:   10000001

-----
Answer: -127 = 10000001

-----
Enter decimal value:
```

Code:

```
def binaryToDecimal(val: str) -> int:
    return int(val, 2)

def decimalToBinary(n: int) -> str:
    return bin(n).replace("0b", "")
```

```
def fillBinaryZero(binary: str, bitLenght: int) -> str:
```

```
    offset = bitLenght - len(binary)
```

```
    toReturn = binary[::-1]
```

```
    for i in range(0,offset):
```

```
        toReturn += '0'
```

```
    return toReturn[::-1]
```

```
def twosComplement(binary: str, isNegative: bool) -> str:
```

```
    # 1. Convert the positive value into 7-bit binary
```

```
    # remove MSB
```

```
    reversedStrBinary = binary[1:][::-1]
```

```
    print('removesignbit: ', reversedStrBinary[::-1])
```

```
    # Invert
```

```
    # 2. Invert the digits (12 -> 02, 02 -> 12)
```

```
    invertedStrBinary = ""
```

```
    for i in range(0,len(reversedStrBinary)):
```

```
        invertedStrBinary += '0' if reversedStrBinary[i] == '1' else '1'
```

```
    print('\n-----')
```

```
    print('2. Inverting the bit')
```

```
    print('invert binary: ', invertedStrBinary[::-1])
```

```
    # add 1s
```

```
    # 3. Add 1s complement
```

```
    if '1' in invertedStrBinary:
```

```
        invertedStrBinary = fillBinaryZero(decimalToBinary(binaryToDecimal(invertedStrBinary[::-1]) + 1),
len(reversedStrBinary))
```

```
    else:
```

```
        invertedStrBinary = invertedStrBinary[0:-1] + '1'
```

```
    print('\n-----')
```

```
    print('3. Add 1 complement')
```

```
    print('+1 Binary: ', invertedStrBinary)
```

```
    # 4. Add 1s sign bit to negative
```

```
    print('\n-----')
```

```
    print('4. Apply negative sign to the MSB')
```

```
    return '1' + str(invertedStrBinary)
```

```

def main() -> None:
    """ Description """

    while True:
        userInput = input("Enter decimal value: ")

        # Without any input, break the loop
        if len(userInput) == 0:
            print("End of program.")
            break

    try:
        while True:
            bitLenght = int(input("Enter Total Bit Lenght (including sign bit): "))
            break
    except:
        print("Invalid bit lenght")
    decimal = int(userInput)

    if bitLenght - len(userInput) < 0:
        print('Invalid Bit Lenght')
    else:
        # check if negative
        if decimal < 0:
            isNegative = True
            decimal = decimal * -1
        else:
            isNegative = False

    binary = fillBinaryZero(str(decimalToBinary(decimal)), bitLenght)
    print('\n-----')
    print('1. Convert to Binary\nIf negative decmial, change to positive decimal into 7-bit binary first')
    print(f'Decimal: {userInput:>14}\n+ve Decimal: {decimal:>10}\nBinary: {binary:>15}')
    finalBinary = twosComplement(binary,isNegative) if isNegative else binary
    print('2scomplement: ', finalBinary)
    print('\n-----')
    print(f'Answer: {userInput} = {finalBinary}')
    print('-----\n')

```

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

Inheritance

```
class Person(object):

    # Constructor
    def __init__(self, name):
        self.name = name

    # To get name
    def getName(self):
        return self.name

    # To check if this person is an employee
    def isEmployee(self):
        return False

# Inherited or Subclass (Note Person in bracket)
class Employee(Person):

    # Here we return true
    def isEmployee(self):
        return True

emp = Person("Geek1") # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

Create SQLite DB from Wikipedia

Create an SQLite DB based on the British Monarch Family Tree.

https://en.wikipedia.org/w/index.php?title=Family_tree_of_British_monarchs&oldid=1043575587

```
## set up tables: british_monarch_family_tree
import sqlite3
conn = sqlite3.connect('british_monarch_family_tree.db')

cur = conn.cursor()

cur.execute('DROP TABLE IF EXISTS british_monarch_family_tree ')
cur.execute('CREATE TABLE british_monarch_family_tree (\
    'id INTEGER PRIMARY KEY AUTOINCREMENT, \
    'name TEXT, \
    'wiki_url TEXT)')

import requests
from bs4 import BeautifulSoup

# Fetch the page content from the url
user_agent = {'User-agent': 'Mozilla/5.0'}
url = 'https://en.wikipedia.org/w/index.php?title=Family_tree_of_British_monarchs&oldid=1043575587'
page = requests.get(url, headers = user_agent)
soup = BeautifulSoup(page.content)

conn = sqlite3.connect('british_monarch_family_tree.db')
cur = conn.cursor()

for tr in soup.find('table').find_all('tr'):
    link = tr.find('a')
    if link and link.text != '' and 'House' not in link.text and 'House' not in link.attrs['href'] and 'History' not in link.attrs['href']:
        cur.execute('INSERT INTO british_monarch_family_tree (name, wiki_url) VALUES (?, ?)',
```

```
(link.get_text(separator=" ").strip(), link.attrs['href']))
conn.commit()
conn.close()
```

Adding foreign key, Mother and Father.

```
# Alter table to add columns with foreign key
conn = sqlite3.connect('british_monarch_family_tree.db')
cur = conn.cursor()
cur.execute('ALTER TABLE british_monarch_family_tree '\
            'ADD COLUMN father_id INTEGER DEFAULT NULL '\
            'REFERENCES british_monarch_family_tree(id)')
cur.execute('ALTER TABLE british_monarch_family_tree '\
            'ADD COLUMN mother_id INTEGER DEFAULT NULL '\
            'REFERENCES british_monarch_family_tree(id)')
conn.close()

# Fetch all from table
user_agent = {'User-agent': 'Mozilla/5.0'}

conn = sqlite3.connect('british_monarch_family_tree.db')
cur = conn.cursor()
cur.execute('SELECT COUNT(*) from british_monarch_family_tree')
table_size = cur.fetchone()[0]

#for c in range(1, table_size+1):
c = 1
while(c <= table_size):
    cur.execute("SELECT id, wiki_url from british_monarch_family_tree WHERE id = ?", (c,))
    r = cur.fetchone()

    # Navigate to each wiki_url https://en.wikipedia.org+wiki_url
    if r[1] == "":
        pass

    url = 'https://en.wikipedia.org' + r[1]
    page = requests.get(url, headers = user_agent)
    soup = BeautifulSoup(page.content)

    # find the father mother
```

```

th_content = soup.find_all(class_="infobox-label")
td_content = soup.find_all(class_="infobox-data")
for i in range(0,len(th_content)):
    if th_content[i].text == 'Father' or th_content[i].text == 'Mother':
        link_tag = td_content[i].find('a')
        if link_tag is None:
            link = ""
        else:
            link = link_tag.attrs['href']
        name = td_content[i].get_text(separator=" ").strip()
        # Check db for existing record of parent and get id
        cur.execute("SELECT id from british_monarch_family_tree WHERE wiki_url LIKE ? OR name LIKE ?",
                    (link,name))
        # Add to db if missing
        if cur.fetchone() is None:
            cur.execute('INSERT INTO british_monarch_family_tree (name, wiki_url) VALUES (?, ?)',
                        (name, link))
            conn.commit()

        # Question did not state to find the parents of the parents
        # If true,row count will be 1000 == DatabaseError: database disk image is malformed
        # table_size += 1

        # Get the father id
        cur.execute("SELECT id from british_monarch_family_tree WHERE wiki_url LIKE ? OR name LIKE ?",
                    (link,name))

        # Update record with parents
        parent_id = cur.fetchone()
        if th_content[i].text == 'Father':
            cur.execute("UPDATE british_monarch_family_tree SET father_id = ? where id=?", (parent_id[0],r[0]))
        elif th_content[i].text == 'Mother':
            cur.execute("UPDATE british_monarch_family_tree SET mother_id = ? where id=?", (parent_id[0],r[0]))

        c+=1
conn.commit()
conn.close()

```

Find all children of King "George III":


```

conn = sqlite3.connect('british_monarch_family_tree.db')
cur = conn.cursor()
cur.execute('SELECT * from british_monarch_family_tree b1 '\
            'WHERE father_id = (SELECT id from british_monarch_family_tree '\
            'WHERE name = "George III")')
result = cur.fetchall()
conn.close()
for r in result:
    print (r)

```

Output:

```

(85, 'George IV', '/wiki/George_IV_of_the_United_Kingdom', 84, 176)
(86, 'William IV', '/wiki/William_IV_of_the_United_Kingdom', 84, 176)
(87, 'Edward Duke of Kent and Strathearn', '/wiki/Prince_Edward,_Duke_of_Kent_and_Strathearn', 84, 176)

```

Find the father and mother of King "George III":

```

conn = sqlite3.connect('british_monarch_family_tree.db')
cur = conn.cursor()
cur.execute('Select * from british_monarch_family_tree '\
            'WHERE id = (SELECT father_id from british_monarch_family_tree WHERE name = "George III") '\
            'OR id = (SELECT mother_id from british_monarch_family_tree WHERE name = "George III")')
result = cur.fetchall()
conn.close()
for r in result:
    print (r)

```

Output:

```

(83, 'Frederick Prince of Wales', '/wiki/Frederick,_Prince_of_Wales', 82, 174)
(175, 'Princess Augusta of Saxe-Gotha', '/wiki/Princess_Augusta_of_Saxe-Gotha', None, None)

```

Find all descendants of "Queen Victoria":

```

# Load db table into df
conn = sqlite3.connect('british_monarch_family_tree.db')
cur = conn.cursor()
cur.execute('SELECT * from british_monarch_family_tree')

```

```

result = cur.fetchall()
conn.close()

df_db = pd.DataFrame(result,
                      columns=['id', 'name', 'wiki_url', 'father_id', 'mother_id'])
df_db = df_db.set_index('id')

# find queen victoria id
root_id = df_db.loc[df_db['name'] == 'Victoria'].index[0]

# make a temp list of descendants to hold index
descendants = []

# insert victoria id into the list index 0
descendants.append(root_id)

# while loop to keep looping till no more descendant
# for loop to fetch new descendants
index = 0
while(index < len(descendants)):
    for id in df_db.loc[(df_db['father_id'] == descendants[index]) |
                      (df_db['mother_id'] == descendants[index])].index:
        if id not in descendants:
            descendants.append(id)
    index+=1

# print df and skip first index 0 victoria
df_db.iloc[[i-1 for i in descendants[1::]]]

```

Output in Dataframe:

id	name	wiki_url	father_id	mother_id
89	Edward VII	/wiki/Edward_VII	178.0	88.0
90	George V	/wiki/George_V	89.0	179.0
91	Edward VIII	/wiki/Edward_VIII	90.0	180.0
92	George VI	/wiki/George_VI	90.0	180.0
93	Elizabeth II	/wiki/Elizabeth_II	92.0	181.0

Pandas

Set max row for dataframe output

```
# max row to display dataframe set to 7
import pandas as pd
pd.set_option('display.max_rows', 7)
```

Combine df with concat

```
# Combine all dataframe with concat  
frame_format = [df_f0_f1, df_f1_f2, df_f2_f3, df_f3_f4, df_f4_f5]  
df_all = pd.concat(frame_format, sort = False)
```

Create histogram from df

df_trace_person_count sample:

	trace id	no of unique person
0	TRACE_PERSON_0000000003	16
1	TRACE_PERSON_0000000005	20
2	TRACE_PERSON_0000000011	5
...
313	TRACE_PERSON_0000000995	20
314	TRACE_PERSON_0000000998	6
315	TRACE_PERSON_0000000999	8

Code:

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10,10))
axes1 = fig.add_subplot(1,1,1)
axes1.hist(df_trace_person_count['no of unique person'],bins=72, color='black')
axes1.set_xlabel('Count of people')
axes1.set_ylabel('Count of clusters')
```

Result:

