programs_1

April 13, 2023

https://www.geeksforgeeks.org/python-programming-examples/

https://github.com/Sachin-D-N/Python_solved_problems/tree/master/Python_Problem_Practise

[1]:
```
import time
import itertools
import numpy as np
from bitarray import bitarray
```

[20]:
```
# sorting of numbers present as dictionary values.
dict_={"hari":29,"mani":30,"siva":29,"ganesh":28,"tirupati":31}

# sorted; returns list
b=sorted(dict_.values())
print( b )
# b => [28, 29, 29, 30, 31]

print( set(b) )
# => {28, 29, 30, 31}

print( list(set(b)) )
# => [28, 29, 30, 31]

_ = b.sort()
print( b )
# => [28, 29, 29, 30, 31]
```

```
[28, 29, 29, 30, 31]
{28, 29, 30, 31}
[28, 29, 30, 31]
[28, 29, 29, 30, 31]
```

[30]:
```
# remove duplicate/copy of word in sentence.
str_="Python is great and Java is also great"

a=str_.split(" ") # it splits and converts the string in to direct list.
print(a)
# => ['Python', 'is', 'great', 'and', 'Java', 'is', 'also', 'great']
```

```
print(dict.fromkeys(a))
# => {'Python': None, 'is': None, 'great': None, 'and': None, 'Java': None,␣
 ↪'also': None}

print( set(a) )
# => {'also', 'Java', 'is', 'Python', 'great', 'and'}

print(" ".join(dict.fromkeys(a)))
```

```
['Python', 'is', 'great', 'and', 'Java', 'is', 'also', 'great']
{'Python': None, 'is': None, 'great': None, 'and': None, 'Java': None, 'also':
None}
{'also', 'Java', 'is', 'Python', 'great', 'and'}
Python is great and Java also
```

[40]:
```
# print the 2nd grade of the students
g_ = 2 # 2nd grade
list_=[["hari",40],["banu",70],["bavani",40],["sachin",80],["sachin",50]]

lt_=[]
for i in list_:
    lt_.append(i[1])

print(lt_)
# => [40, 70, 40, 80, 50]

lt_=set(lt_)
print(lt_)
# => {40, 50, 80, 70}

for i in list_:
    if i[1]==list(lt_)[g_-1]:
        print(i)
```

```
[40, 70, 40, 80, 50]
{40, 50, 80, 70}
['sachin', 50]
```

0.1 What is Lambda Function in Python?

A Lambda Function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.

[42]:
```
# reversing a list using slicing technique
def Reverse(lst):
    new_lst = lst[::-1]
    return new_lst
```

```
lst = [10, 11, 12, 13, 14, 15]
print(Reverse(lst))

# OR ==========================
lst.reverse()
print("Using .reverse() ", lst)
```

```
[15, 14, 13, 12, 11, 10]
Using .reverse()  [15, 14, 13, 12, 11, 10]
```

[12]:
```
# rotate the numbers clockwise.
n=[1,2,3,4,5]
l=int(input("enter the no. of rotations : "))
for i in range(1,l+1):
    # n[-j:] => # returns list of j numbers from the end.
    # n[:j] => # returns list of j numbers from start.
    n = n[-len(n)+1:] + n[:1]
    print("rotation no. ",i, " =>", n)

# check the every element of every list from up to down => feel like clockwise.
```

```
enter the no. of rotations : 7
rotation no.  1  => [2, 3, 4, 5, 1]
rotation no.  2  => [3, 4, 5, 1, 2]
rotation no.  3  => [4, 5, 1, 2, 3]
rotation no.  4  => [5, 1, 2, 3, 4]
rotation no.  5  => [1, 2, 3, 4, 5]
rotation no.  6  => [2, 3, 4, 5, 1]
rotation no.  7  => [3, 4, 5, 1, 2]
```

[10]:
```
# rotate the numbers anti-clockwise.
n=[1,2,3,4,5]
l=int(input("enter the no. of rotations : "))
for i in range(1,l+1):
    n = n[-1:] + n[:len(n)-1]
    print("rotation no. ",i, " =>", n)

# check the every element of every list from up to down => feel like␣
 ↪anti-clockwise.
```

```
enter the no. of rotations : 7
rotation no.  1  => [5, 1, 2, 3, 4]
rotation no.  2  => [4, 5, 1, 2, 3]
rotation no.  3  => [3, 4, 5, 1, 2]
rotation no.  4  => [2, 3, 4, 5, 1]
rotation no.  5  => [1, 2, 3, 4, 5]
rotation no.  6  => [5, 1, 2, 3, 4]
rotation no.  7  => [4, 5, 1, 2, 3]
```

[4]:
```
# right rotating a list by picking n positions from the end.
n = 3
list_1 = [1, 2, 3, 4, 5, 6]

if n>len(list_1):
    n = int(n%len(list_1))# % returns remainder. i.e., 1 % 5 = 1

# list_1[-n:] => # returns list of n numbers from the end.
# list_1[:-n] => # returns list from start, leaving n numbers from the end.
list_1 = list_1[-n:] + list_1[:-n]

print(list_1)
```

```
[4, 5, 6, 1, 2, 3]
```

[23]:
```
# duplicate remove using lists;

l = [1, 2, 4, 2, 1, 4, 5]
print("Original List: ", l)
# type( set(l) ) => <class 'set'>
# set(l) => {1, 2, 4, 5}
# *set(l) => 1 2 4 5
res = [*set(l)]
print("List after removing duplicate elements: ", res)
```

```
Original List:  [1, 2, 4, 2, 1, 4, 5]
List after removing duplicate elements:  [1, 2, 4, 5]
```

[26]:
```
# OR ====================================
mylist = ["hello", 2 ,"hello", 3, 4,6,5,5, 5]

# dict.fromkeys(mylist) => {'hello': None, 2: None, 3: None, 4: None, 6: None,␣
 ↪5: None}
mylist = list(dict.fromkeys(mylist))
print(mylist)
```

```
['hello', 2, 3, 4, 6, 5]
```

[37]:
```
b=[1,2,3,1]
_ = b.sort()
print(b)
# => [1, 1, 2, 3]

del b[0]
_ = b.sort()
print(b)
# => [1, 2, 3]
```

```
_ = b.remove(2) # remove the number '2' from the list.
print(b)
```

```
[1, 1, 2, 3]
[1, 2, 3]
[1, 3]
```

[1]:
```
# number of solutions to modular equations.
# (a % x) = b => % (modulo operator)

a=int(input("element..."))
b=int(input("modulus..."))
print('')
l=[]
for x in range(1,a):
    if a%x==b:
        print("modulus by...",x)
        l.append(x)
print("possible values of x ...",l)
```

```
element…26
modulus…2

modulus by… 3
modulus by… 4
modulus by… 6
modulus by… 8
modulus by… 12
modulus by… 24
possible values of x … [3, 4, 6, 8, 12, 24]
```

[9]:
```
# Check whether a "decimal number" has consecutive 0's in the given base-form␣
 ↪or not.

n=int(input("enter the number ..."))
k=int(input("enter the base ..." ))

if k==2:
    # bin(n) => binary of n.
    # bin(16) => 0b10000
    # type( bin(16) ) => <class 'str'>
    b=bin(n)[2::] # select characters on or after 2 in string.
    # b => 10000

    # b.count('0') => count 0's in string 'b'.
    print("count of zeros in base-form ({}) is {}".format(k, b.count('0')))
```

```
elif k==8:
    # hex(16) => 0x10
    b=hex(n)[2::]
    print("count of zeros in base-form ({}) is {}".format(k, b.count('0')))

elif k==16:
    b=oct(n)[2::]
    print("count of zeros in base-form ({}) is {}".format(k, b.count('0')))
else:
    print("wrong k notation")
```

```
enter the number …256
enter the base …16
count of zeros in base-form (16) is 2
```

[13]:
```
# given a number n, find whether all digits of n divide it or not.
def digit_divisible(n):
    s=str(n)
    flag=True
    for i in s:
        if n%int(i)==0:
            continue
        else:
            flag=False
    if flag:
        return 'All the digits of number can divide the number', n
    else:
        return 'All the digits of number cannot divide the number',n

digit_divisible(int(input('Enter the number-->')))
```

```
Enter the number-->128
```

[13]:
```
('All the digits of number can divide the number', 128)
```

[26]:
```
# replace duplicate/copy from string leaving first one i.e, from second␣
 ↪occurrence.

test_str = 'India is best . India has many states . India can help other␣
 ↪countries to grow.'

# initializing replace mapping.
repl_dict = {'India' : 'It', 'other' : 'rest of' }

test_list = test_str.split(' ')
# test_list =>
```

```
# ['India', 'is', 'best', '.', 'India', 'has', 'many', 'states', '.', 'India',␣
 ↪'can', 'help', 'other', 'countries', 'to', 'grow.']


# list comprehension offers a shorter syntax to complete all with only one line␣
 ↪of code.
# test_list.index(val) => get index from "test_list" list where value is "val".
res = ' '.join([repl_dict[val] if val in repl_dict.keys() and test_list.
 ↪index(val) != idx else val for idx, val in enumerate(test_list)])

# printing result
print("The string after replacing ===> " + str(res))
```

```
The string after replacing ===> India is best . It has many states . It can help
other countries to grow.
```

[25]:
```
# replace multiple words with K.

a="my name is kavya my father name is x"
b=["my","name","is"]
for i in b:
    a=a.replace(i,"hi")
print(a)
```

```
hi hi hi kavya hi father hi hi x
```

[196]:
```
%%time
# list all primes number below n
def bit_primes(n):
    # 30 // 3 => 10
    # 30 % 6 == 2 => False
    # 30 // 3 + (30 % 6 == 2) => 10
    # 10 + False, 10 + True => 10, 11
    # bitarray( 10 ) => creates an random array of bits of size 10.
    bit_sieve = bitarray(n // 3 + (n % 6 == 2))
    # bit_sieve => bitarray('0000000000')
    # type(bit_sieve) => <class 'bitarray.bitarray'>

    bit_sieve.setall(1)
    # bit_sieve => bitarray('1111111111')

    bit_sieve[0] = False
    # bit_sieve => bitarray('0111111111')

    # 30 ** 0.5 => 30 power 0.5.
    # int(30 ** 0.5) // 3 + 1 => 2
    for i in range(int(n ** 0.5) // 3 + 1):
```

```
        if bit_sieve[i]:
            """
            The | (OR) operator performs bit-wise addition (1+1=1, 0+1=1,␣
 ↪0+0=0):

            6 | 3 => 7
            6 = 0000000000000110
            3 = 0000000000000011
            --------------------
            7 = 0000000000000111
            ====================

            Decimal numbers and their binary values:
            0 = 0000000000000000
            1 = 0000000000000001
            2 = 0000000000000010
            3 = 0000000000000011
            4 = 0000000000000100
            5 = 0000000000000101
            6 = 0000000000000110
            7 = 0000000000000111
            """
            k = 3 * i + 1 | 1 # OR operation is performed after BODMAS.
            # k => 5
            bit_sieve[k * k // 3::2 * k] = False
            # bit_sieve => bitarray('0111111101')
            bit_sieve[(k * k + 4 * k - 2 * k * (i & 1)) // 3::2 * k] = False
            # bit_sieve => bitarray('0111111101')

    # bit_sieve.tobytes() => b'\x7f@'
    # np.frombuffer(bit_sieve.tobytes(), dtype=np.uint8) => [127  64]
    # np.unpackbits(np.frombuffer(bit_sieve.tobytes(), dtype=np.uint8)) => [0 1␣
 ↪1 1 1 1 1 0 1 0 0 0 0 0 0]
    np_sieve = np.unpackbits(np.frombuffer(bit_sieve.tobytes(), dtype=np.
 ↪uint8)).view(bool)
    # np_sieve => [False  True  True  True  True  True  True  True False  True␣
 ↪False False False False False False]
    # np.flatnonzero(np_sieve) => [1 2 3 4 5 6 7 9]
    # 3 * np.flatnonzero(np_sieve) + 1 | 1 => [ 5  7 11 13 17 19 23 29]
    return np.concatenate(((2, 3), (3 * np.flatnonzero(np_sieve) + 1 | 1)))

print(bit_primes(30))
```

```
[ 2  3  5  7 11 13 17 19 23 29]
Wall time: 0 ns
```

[ ]:

[223]:
```
%%time
# prime factors of a number.
def prime_factors(n):
    i = 2
    factors = []
    while i * i <= n:
        # different values taken by n % i => 0 0 0 1
        if n % i:
            i += 1
        else:
            n //= i
            factors.append(i)
    if n > 1:
        factors.append(n)
    return factors

print(prime_factors(64))
```

```
[2, 2, 2, 2, 2, 2]
Wall time: 0 ns
```

[5]:
```
# Check if one string is a rotation of other string

def is_rotation(s1, s2):
    # 2*s2 => tackoverflowstackoverflows
    return len(s1) ==len(s2) and s1 in 2*s2

s1 = "stackoverflow"

# s2 = "stackoverflwo" # Should return False
s2 = "tackoverflows" # Should return True
# s2 = "ackoverflowst" # Should return True
# s2 = "overflowstack" # Should return True

print(is_rotation(s1, s2))
```

```
True
```

[13]:
```
%%time
# Find "Sum of Digits" / "Digital Root".
# Digital root is the recursive sum of all the digits in a number.
# 493193  -->  4 + 9 + 3 + 1 + 9 + 3 = 29  -->  2 + 9 = 11  -->  1 + 1 = 2

def digital_root(n):
    if(n < 10):
        return n
```

```
    """
        n = 235%10 + digital_root(235//10)
          = 5 + digital_root(23)
                        |
                n = 23%10 + digital_root(23//10)
                  = 3 + digital_root(2)
                                |
                            return 2
                  = 5
                return digital_root(5)
                              |
                            return 5
                return 5
          = 5 + 5
          = 10

        return digital_root(10)
                     |
                n = 10%10 + digital_root(10//10)
                  = 0  + digital_root(1)
                                |
                             return 1
                  = 1
                return digital_root(1)
                              |
                            return 1
                return 1
        return 1
    """

    n=n%10+digital_root(n//10) # remainder + digital_root(quotient)
    return digital_root(n)

print(digital_root(235))
```

```
1
Wall time: 0 ns
```

[19]:
```
# OR ====================================
# without recursion
def root(n):
    while n > 9:
        # map(int, str(n)) => <map object at 0x0000027A5CB1C848>
        # list(map(int, str(n))) => [2, 3, 5]
        n = sum(map(int, str(n)))
        # n => 10
    return n
```

```
print(root(235))
```

```
1
```

[23]:

```

```

```
UP 5 DOWN 3 LEFT 3 RIGHT 2
```

```
5
```

[ ]:

```

```