```
# odwróć listę, tzn. zamień np. [3,4,5] na [5,4,3]
return list(reversed(L))

plik = open('popularne_slowa.txt', 'r')  # 'r' -- czytanie, można pominąć, 'rb' -- czytanie binarnego

tekst = plik.read().split()
#wiersze_pliku = plik.readlines() # alternatywa

wiersz = wiersz.rstrip() # usuń białe znaki z końca

a.add(10) # dodanie elementu do zbioru
a.append(10) # dodanie elementu do listy

a | b  # suma zbiorów
a & b  # część wspólna zbiorów
a <= b  # zawieranie zbiorów
len(set(L)) # liczba unikalnych elementów w liście
sorted(L) # nie zmienia listy L
L.sort()  # zmienia
```

```
i[0].lower()
```

```
if type(l)==str:
print(' '.join(L))
```

```
#szyfr cezara
litery="aąbcćdeęfghijklłmnńoóprsśtuwyzźż"

def zaszyf(n,k):
    w=""
    k=k%32
    for i in range(len(n)):
        j=0
        while n[i]!=litery[j]:
            j=j+1
        w=w+litery[(j+k)%32]
    return w

def odszyf(n,k):
    w=""
    k=k%32
    for i in range(len(n)):
        j=0
        while n[i]!=litery[j]:
            j=j+1
        w=w+litery[(j-k+32)%32]
    return w

def czy_szfr(n,s):
    i=0
    while n[0]!=litery[i]:
        i=i+1
    j=0
    while s[0]!=litery[j]:
        j=j+1
    k=(j-i+32)%32
    if zaszyf(n,k)==s:
        return True
    else:
        return False
```

```
def kwadrat(kolor):
    begin_fill()
    colormode(255)
    fillcolor(kolor)
    pencolor(kolor)
    for i in range(4):
        fd(bok)
        rt(90)
    end_fill()

move(0,0)
j=0
for wiersz in open('niespodzianka.txt').readlines():
    move(0,200-bok*j)
    L=wiersz.split()
    for i in range(len(L)):
        L[i]=eval(L[i])
    for k in L:
        kwadrat(k)
        fd(bok)
    j=j+1
```

```
from random import *
from turtle import *
import numpy as np

tracer(0,0)

...

update()
input()
```

```
random.random()
```

```
from random import choice

fragmenty = "s...ziu".split()

def losuj_fragment():
    return choice(fragmenty)
```

```
print(sorted(r,key= lambda x:r[x],reverse = True)) #wypisuje klucze slownika r w kolejnosci posortowanej wedlug wartosci malejaco
```

```
def slownik(slowo):
    literki={}
    for l in slowo:
        if l not in literki:
            literki[l]=0
        literki[l]=literki[l]+1
    return literki

def czy_ukladane(w,h):
    word=slownik(w)
    hand=slownik(h)
    return all(x if x in hand.keys() and hand[x]>=word[x] else False for x in word.keys())
```

```
def fib(n):
    if n <= 1:
        return 1
    return fib(n-1) + fib(n-2)

for i in range(40):
    print (fib(i))
```

```
def krzyzyk(bok, kolor):
    begin_fill()
    kolor=[float(x) for x in kolor]
    fillcolor(kolor)
    rt(45)
    for i in range(4):
        rt(90)
        fd(bok)
        rt(90)
        fd(bok)
        lt(90)
        fd(bok)
    end_fill()
```

```
colors = {
  'r' : 'red',
  'g' : 'green',
  'b' : 'blue',
  'y' : 'yellow',
  'o' : 'orange',
}

def kwadrat(kolor):
    begin_fill()
    fillcolor(colors[kolor])
    for i in range(4):
        fd(30)
        rt(90)
    end_fill()
```

```
def permutacja(L):
    L = L[:]
    random.shuffle(L)
    return tuple(L)
```

```
random.randint(1,6)
```

```
def randperm(n):
    return random.sample(range(0,n),k=n)
```

```
def usun_duplikaty(L):
    wynik=[]
    for e in L:
        if e not in wynik:
            wynik.append(e)
    return wynik
```

```
mieszanka = (1-alfa) * kolor1 + alfa * kolor2
```

```
for s in pol:
    o=wzaj_odwr(s)
    if o in pol:
        if s<o:
            print(s,o)
```

1. range(od ile, do ile, co ile) jeśli range(10) to domyślnie [0,10) co 1
2. znak konca wiersza '\n'
3. circle(promien,-kacik) jeśli bez kata powstanie kolo
4. pass, break, continue

```
def pierwsza(n):
    if n==1:
        return False
    if n==2:
        return True
    for i in range(3,(n**(0,5)),2):
        if n%i==0:
            return False
    return True
```

```
kolor1=np.array([randrange(0,10)/10,randrange(0,10)/10,randrange(0,10)/10])
```

```
from collections import defaultdict as dd
pusta_lista = []
pol_ang = dd(lambda : pusta_lista)   # list    lambda:[]
```

```
tracer(0,1)
print ('Zmienna __name__ =', __name__)

if __name__ == '__main__':
    speed('fastest')
    kolory = ['red', 'green', 'blue']

    for i in range(10):
        kwadrat(i,i, kolory[i % 3])

    input()
```

```
def perm(L):
    if len(L) == 0:
        return [ [] ]  # a co gdyby: []
    ps = perm(L[1:])
    e = L[0]
    return [ p[:i] + [e] + p[i:]  for p in ps for i in range(len(p)+1) ]
```

```
def jest_cyfra(d):
    return d.isdigit()
```

```
abs(x) #wartosc bezwzgledna z x
tuple() #krotka
eval(f, {'x' : x, 'sin' : math.sin, 'cos' : math.cos})
ord(d) #kod ascii
```

```
for i in range(10):
    x = random.randint(1, 10)
    try:
        print (f'1 / {x} == {1/xyz}')
    except ZeroDivisionError:
        print('Dzielenie przez zero?')
    except NameError:
        print ('Zła nazwa')
```

```
def safe_int(d):
    try:
        return int(d)
    except ValueError:
        return None
```

```
class Set:
    def __init__(self, *elems):
        self.tree = []
        for e in elems:
            self.add(e)

    def add(self, e):
        add_to_tree(e, self.tree)

    def __contains__(self, e):
        return in_tree(self.tree, e)

    def __str__(self):
        return f'Set({tree_to_list(self.tree)})'

    def __or__(self, other):
        new = Set(*tree_to_list(self.tree))
        for e in tree_to_list(other.tree):
            new.add(e)
        return new

zbior = Set(1,4,5,3,3,5,1,1,1,8)

print(zbior)

for i in range(7):
    print (i, i in zbior)

print (Set(1,2,3,4,4,4) | Set(3,44,4,4,5))
```

```
import random
chłopcy = ['Janek', 'Piotr', 'Adaś', 'Brajan', 'Artur']
dziewczyny = ['Ania', 'Basia', 'Celina', 'Dżessika', 'Ewelina', 'Nikol']
czapeczki = ['melonik', 'beret', 'bejsbolówka', 'kaszkiet']  # brakuje jednej czapki!

def narzednik(x):
    if x[-1] == 'a':
        return x[:-1] + 'ą'
    return x

random.shuffle(chłopcy)
random.shuffle(dziewczyny)

for c,d,cz in zip(chłopcy, dziewczyny,czapeczki):
    print (f'{c} będzie tańczyć poloneza z {narzednik(d)}, a na głowie {cz}.')
```

```
#obliczyć listę podzbiorów listy różnych elementów L
def subsets(L):
    if len(L) == 0:
        return [set()]
    ss = subsets(L[1:])
    return ss + [{L[0]} | s for s in ss]
```

```
class Osoba:
    def __init__(self, im, naz, w):
        self.imie = im
        self.nazwisko = naz
        self.wiek = w

    def plynie_czas(self):
        self.wiek += 1

    def __str__(self):
        return f'osoba({self.imie}, {self.nazwisko})'

    def __repr__(self):
        return f'Osoba("{self.imie}", "{self.nazwisko}", {self.wiek})'

    def __hash__(self):
        return hash(  (self.imie, self.nazwisko, self.wiek) )

    def __eq__(self, other):
        return self.imie == other.imie and self.nazwisko == other.nazwisko and self.wiek == other.wiek
```

```
janek = Osoba('Jan', 'Kowalski', 34)
ala = Osoba('Alicja', 'Nowak', 22)
basia = Osoba('Barbara', 'Pisarska', 12)

osoby = [janek, ala, basia]

print ('Raport 1')
for o in osoby:
    print (o.imie, o.nazwisko, 'ma', o.wiek, 'lat')
print ()

for o in osoby:
    o.plynie_czas()

print ('Raport 2')
for o in osoby:
    print (o, o.__str__())
print ()

print (osoby)

dziewczyny = {ala, basia}
dziewczyny.add(Osoba('Celina', 'Nowak', 20))

print (dziewczyny, ala in dziewczyny)
print (len({Osoba('Alicja', 'Nowak', 22), ala}))
```

```
def in_tree(tree, e):
    if tree == []:
        return False
    n, left, right = tree
    if n == e:
        return True
    if e < n:
        return in_tree(left, e)
    return in_tree(right, e)

def tree_to_list(tree):
    if tree == []:
        return []
    n, left, right = tree
    return tree_to_list(left) + [n] + tree_to_list(right)

def add_to_tree(e, tree):
    if tree == []:
        tree += [e, [], []]
        return
    x, left, right = tree
    if e < x:
        add_to_tree(e, left)
    elif e > x:
        add_to_tree(e, right)
```