---sprawdzamy czy to jest ciąg arytmetyczny:

if input[1] - input[0] == input[2] - input[1]:

print(True)

else:

print(False)

---sprawdzamy tuplę, trzeba najpierw zrobić jej split:

list = input[0].split() # splitujemy input[0]

if int(list[1]) - int(list[0]) == int(list[2]) - int(list[1]):

print(True)

else:

print(False)

----zadanie: sprawdź czy to są ciągi:

(moze też być lista lub tupla: input = ("-85 -177 -269", "-80 -20 66", "42 89 136", ...)

for i in input: # i to input[0], input[1]...

list = i.split()

if int(list[1])-int(list[0]) == int(list[2])-int(list[1]):

print True

else:

print False

JEŚLI ROBISZ SPLIT TO ZAWSZE TRZEBA WRZUCIĆ WYNIK DO LISTY!!!!

----sprawdzamy czy to jest ciąg geometryczny (mnożony przez tę samą liczbę, np x2: 2,4,6... albo 5,10,15.. albo x3: 2,6,24,72...

input = ["2 4 6 8", "12 45 56 78 1 3 56", "3 6 9 18 36"] # lista o różnej długości ciągach-stringach

for i in input:

list = i.split()

wynik = True

dzielnik = float(list[1])/float(list[0]) # ustalamy o ile mnożone są kolejne liczby ciągu

for j in range (len(list)-2):

if dzielnik != (float(list[j+2])/float(list[j+1])):

wynik = False

break

print wynik

---pixblocks mają len(list)-1 i float(list[j+1])/float(list[j]) no i mają rację bo w sumie i tak pętla sprawdza wszystkie elemanty ciągu porównuje do dzielnika.

----wylicz następny elememt ciągu arytmetyczego:

input = ("-81 -87 -93 -99", "-80 -59 -38 -17 4 25", )

for i in input:

list = i.split()

roznica = int(list[1])-int(list[0])

last = int(list[len(list)-1] ) # można zastąpić list[-1]

next = last + roznica

print next

---pixblox ma tak samo, tylko nie rozpisuje równania na zmienne "różnica" i "last" (jak ja dla czytelności) i wrzuca równanie od raz do print, oraz zamiast szukać ostatniego wyrazu ciągu sparawdzając jego długość, to ma: list[-1]:

list[-1] <<= pokazuje ostatni elememt listy!!!!!

for i in input:

list = i.split()

print (int(list[-1])+int(list[1])-int(list[0]))

wynik:

105

54

---wypisz 3 następne wyrazy ciągu arytmetycznego

input = („123 125”, „246 286”)

for i in input:  
 a,b = float(i.split()[0]),float(i.split()[1])  
 dzielnik = b/a  
 lista = []  
 x = b  
 for i in range (3):  
 x\*=dzielnik  
 lista.append(str(x))  
   
 wynik = "" # musi być tu a nie przed pętlą for, bo inaczej nie wyjdzie  
 for j in lista: # ta pętla zamienia znowu w string ze spacjami  
 wynik += j + ' '  
 print (wynik)

----jeśli ciąg jest arytmetyczny to wypisz jego następny element, inaczej wypisz False:

input = ("123 125 127", "23 34 56 67 78", "23 4 67")

for i in input:

list = i.split()

roznica = int(list[1]) - int(list[0]) # wyświetla łąd, bo nie ma "in range"!!!

for j in **range**(len(list)-1):

if (list[j+1]-list[j])==roznica:

print (list[-1] + roznica)

else:

print False

break

--poprawki:

input = ("123 125 127", "23 34 56 67 78", "23 4 67")

for i in input:

arytmetyczny = True

list = map(int,i.split())

r = (list[1]) - (list[0])

for j in range(len(list)-1):

if (list[j+1]-list[j])!= r:

arytmetyczny = False

break

#(nie ma sensu robić else bo i tak arytmetyczny = True)

if arytmetyczny == True:

print (int(list[-1])+r)

else:

print (False)

----pix:

for i in input:

list = i.split()

is\_arithmetic = True

r = int(list[1]) - int(list[0])

for j in range(len(list)-1):

if(r != (int(list[j+1]) - int(list[j]))):

is\_arithmetic = False

break

if(is\_arithmetic == True):

print (int(list[-1])+int(list[1])-int(list[0])) # tutaj "r" zamiast

else:

print (False)

129

False

False

---3 liczby - pierwsza to pierwszy el. ciągu, druga to wielkość skoku, a trzecia to indeks elementu, który należy obliczyć:

iput = ["-11 -6 19", "5 11 11", "8 -5 36"]

for i in input:

dane = i.split()

c = int(dane[0])

for j in range (int(dane[2])-1):

c+=int(dane[1])

print (c)

wynik:

-119

115

-167

----sprawdz czy ciągi są arytmetyczne i wypisz ich monotoniczność: jeśli są rosnące, to napisz "increasing", malejące "Decreasing" a stałe "Constans". Jeśli nie są arytmetyczne to napisz False.

moje (musiałam poprawić tylko tego if-a

for i in input:

list = i.split()

roznica = int(list[1]) - int(list[0])

arytmetyczny = True

for j in range (int(len(list) - 1)):

if (roznica != (int(list[j+1]) - int(list[j]))):

arytmetyczny = False

break

if (arytmetyczny == True): # to musi być na tej samej wys. co "for", bo nie wyjdzie

if ( roznica>0 ):

print ("Increasing")

elif ( roznica<0 ):

print ("Decreasing")

elif ( roznica==0 ):

print ("Constant")

else:

print False

---oblicz sumę pierwszych n wyrazów ciągu. 1sza liczba to 1szy wyraz ciągu, 2ga l. to wartość skoku, 3cia to n.

for i in input:

list=i.split()

pierwszy = float(list[0])

sum = float(list[0])

for j in range(float(list[2])-1):

pierwszy += float(list[1])

sum += pierwszy

print sum

---pix przekombinował (wartość pierwszego a1 i ostatniego wyrazu ciągu an dzielona na 2 (uśredniona) razy n:

for i in input:

list = i.split()

a1 = float(list[0])

r = float(list[1])

an = float(list[0])

for j in range(int(list[2]) - 1):

an += int(list[1])

sum = ((a1 + an) / 2) \* int(list[2])

print(sum)

----sprawdz czy ciąg jest naprzemianny (raz ujemny raz dodatni, czyli iloczyn kolejnych dwóch wyrazów ciągu powinien być ujemny):

for i in input:

list = i.split()

a1=list[0]

wynik = True

for j in range(int(len(list))-1):

if (int(list[j])\*int(list[j+1])<0):

wynik = True

elif (int(list[j])\*int(list[j+1])>=0):

wynik = False

break

print wynik

---wg. pixblocks (moje znów lepsze):

for i in input:

sequence = i.split()

sequence = map(int, sequence) # chyba nadaje int wszystkim wyrazom listy

isAlternating = True

for j in range(1, len(sequence)):

if sequence[j - 1] == 0 or sequence[j] == 0:

isAlternating = False

break

elif sequence[j - 1] < 0 and sequence[j] < 0:

isAlternating = False

break

elif sequence[j - 1] > 0 and sequence[j] > 0:

isAlternating = False

break

print(isAlternating)

---oblicz ile pieniędzy zarobisz, zaokrąglij do 2 miejsca po przecinku - pierwsza liczba to kapitał początkowy, druga oprocentowanie roczne, trzecia to ile lat trzymane pieniądze:

input = ("350 8% 9", "450 7% 6", "100 15% 3")

for i in input:

ciag= i.split()

a1 = int(ciag[0])

proc = ciag[1]

for j in range (int(ciag[2])):

if ("%" in proc):

new = proc.split("%") # powstaje lista(!!) "new", nie zmienna

new = int(new[0]) \*0.01 # dlatego trzeba wpisać new[0]

a1 = a1+ a1\*new

print (round(a1,2))

--pix, podobnie ale... (nie wiem co to jest to [:-1], może usuwanie ostatniego znaku ze stringa z %:

for i in input:

start = float(i.split()[0])

percentage = float(i.split()[1][:-1])

years = float(i.split()[2])

while years > 0:

start += start \* percentage / 100.0

years -= 1

print(round(start, 2))

---pętla while: wypisz liczby od 20 do 5:

x = 20

while (x >=5):

print(x)

x-=1

---wypisz napis 8 razy:

i = 0

while (i < 8):

print('PixBlocks')

i+=1

---znajdź największa potęgę dwójki mniejszą niż podana liczba:

i = 1

while (i < input[0]):

i \*= 2

print(i/2) # też dzielą na 2

--moje pomyliłam z potęgowaniem i nie wychodziło:

x = 2

while (x < input[0]):

i =i\*\* 2 # 2\*2 potem 4\*4, 16\*16 zamiast 1\*2, 2\*2, 4\*2 itd...

print(i)

---za pomocą pętli while wypisz kolejne potęgi liczby 3, tyle ile podane w input[0]:

i, n = 0, 3

while (i< input[0]):

n\*=3

i+=1

print(n/3)

-- wystarczy zmianić n na 1 i nie trzeba dzielić wyniku przez 3:

i, n = 0, 1

while (i< input[0]):

n\*=3

i+=1

print(n)

---pix, zaczynają od i=1 a nie 0 i dlatego i<= a nie i<:

i, n = 1, 3

while (i <= input[0]):

print(n)

n \*= 3

i += 1

---za pomocą **break** wychodzimy z pętli:

i = 1

while i < 10:

print(i)

if i == 6:

break

i += 1

(wynik: 1,2,3,4,5,6)

---przykład - za pomoca "**continue**" możemy zatrzymać bieżącą iterację i iść do następnej. Dlatego i+=1 musi być wcześniej, bo inaczej powstanie niekończąca się pętla: spełniony if więc continue-wraca do poprzedniego a tam nic się nie zmieniło, więc znów idzie do if:

i = 1

while (i < 10):

i += 1

if i == 6:

continue

print(i)

(wynik: 2,3,4,5,7,8,9,10 --- bez 6)

---wypisz liczby od 1 do 100 pomijając 30 i 45:

x=0

while x<100:

x+=1

if (x==30) or (x==45):

continue

print(x)

----moje:

i=1

while i<=100:

if (i!=30 and i!=45):

print i

else:

i+=1

continue

i+=1

---z for:

for i in range (1,101):

if (i == 30 or i == 45):

continue

print(i)

----break przerwie wypisywanie liczb przy 29 :

i = 0

while (i < 100):

i += 1

if (i == 30 or i == 45):

break

print(i)

----wypisz liczby z podanego zakresu obustronnie zamkniętego, pomijając te podzielne przez 7. Przy 49 zatrzymaj.

input = ["40 50"] #lista z jednym argumentem "40 50" o indeksie 0 (czyli input[0])

list = input[0].split()

i=int(list[0])-1

while (i<int(list[1])):

i+=1

if(i==49):

break

if(i%7==0):

continue

print(i)

(wynik: 40,41,43,44,45,46,47,48)

----pix podobnie tylko zamienił pierwszą i ostatnią cyferkę stringa w zmienne start i end:

start, end = int(input[0].split()[0]) - 1, int(input[0].split()[1])

while start < end:

start += 1

if start % 49 == 0:

break

elif start % 7 == 0:

continue

print(start)

----przykład: else uruchamia blok kodu, który działa gdy warunek przestaje być spełniony:

i = 1

while i < 10:

print(i)

i += 1

else:

print("i is no longer less than 10")

(wynik: 1,2.....8,9,"i is no longer less than 10")

---gdy użyjemy break, else już sie nie wykona:

i = 1

while i < 10:

print(i)

i += 1

if(i==5):

break

else:

print("i is no longer less than 10")

(wynik: 1 2 3 4)

---sprawdź czy pierwszy element listy się powtarza, jeśli tak to wypisz jego indeks i przerwij program, jeśli nie to wypisz "Not found":

first = int(input[0])

i=0

while i<len(input):

i+=1

if (int(input[i])==first):

print (i)

break

else:

print("Not found")

--pix - nie ma potrzeby zamiany na int, bo w input są już liczby, ale rypnęli się, bo dali i=2 i wtedy nie znajdzie pierwszego powtórzenia, np. tutaj wypisze index 7 a nie 1 input = [1,1,2,3,4,5,6,1,2]:

i = 2 # błąd, powinno być i=1

while(i<len(input)):

if (input[0] == input[i]):

print (i)

break

i += 1

else:

print('Not found')

---wypisz tyle liczb pierwszych ile pokazuje podana liczba:

pix:

counter, number = 0, 2

while counter < input[0]:

for i in range(2, number + 1):

if number % i == 0:

if number == i:

print(number)

counter += 1

number += 1

break

---moje:

def prime(x):

for i in range(2,x):

if x%i == 0:

return False

return True

i,j = 2,input[0]

while j>0:

if prime(i):

print (i)

j-=1

i+=1

---wypisz liczby pierwsze mniejsze niż podana liczba (input). PS. w pix input to zawsze lista, dlatego nawet jeśli ma 1 element to dajemy input[0]:

generalnie wszystko jak wyżej, tylko counter(tutaj whi) dodaje +1 za każdym razem a nie tylko gdy drukuje liczbę pierwszą więc wypisze tylko liczby pierwsze mniejsze od podanej a nie tyle liczb piewszych ile podana.

input = [20]

liczba = 2

whi = 0

while whi < input[0]:

for i in range (2,liczba+1):

if liczba % i == 0:

if liczba == i:

print (liczba)

whi+=1

liczba +=1

break

---------znajdź najmniejszy współny dzielnik, wykorzystując metode Euklidesa:



---moje:

for i in input:

a = int(i.split('\t')[0])

b = int(i.split('\t')[1])

while a!=b:

if (a>b):

a=a-b

else:

b=b-a

print (a)

--------pix, ewidentnie wykorzystali inny diagram:



----dokładnie przepisany działa idealnie:

for i in input:

a = int(i.split()[0])

b = int(i.split()[1])

while (b != 0):

reszta = a % b

a = b

b = reszta

print(a)

----wg. pix, troszkę inaczej:

for i in input:

a = int(i.split()[0])

b = int(i.split()[1])

while (b != 0):

temp = b

b = a % b

a = temp

print(a)

---wypisz ciąg Fibonaciego tyle razy ile pokazuje input[0]:

input=['6']

i = 0

print(1)

a=0

b=1

while i < int(input[0])-1

w = a+b

a=b

b=w

print (w)

i+=1

(wynik: 1,1,2,3,5,8,13.....)

--wg pixblocks:

input = [6] # tu nie ma stringa

a, b = 0, 1

while (input[0] > 0): # nie trzeba zamieniać w int

print(b)

b += a

a = b - a

input[0] -= 1

---# czy podana liczba jest l. pierwszą:

def prime(x):

prime = True

if x<2:

prime = False

for i in range(2,x):

if x%i == 0:

if x == i:

prime = True

else:

prime = False

print x, prime

----powyższe coś nie do końca działa, lepiej spróbować takiej wersji:

def prime(x):

if x<2: # (ten pierwszy akapit z if-em jest chyba nawet cały zbędny)

return False # jakby go nie było to by returnował None zamiast False

for i in range(2,x**+1**):

if x%i == 0:

if x == i:

return True

else:

return False

print x, prime

# sparawdzenie liczb od 0 do 20

for i in range (21):

prime(i)

(wynik:

0 False

1 False

2 True

3 True

....

19 True

20 False

--ciekawostka: jeśli w funkcji dam print (x,prime) zamiast print x, prime to wniki też będą w nawiasach:

(0, False)

(1, False) ...

ale jeśli dam print (x), (prime) to nic się nie zmieni, czyli np. 0 False

---Stringi----

---metoda count liczy ile mamy np "is" w stringu **wyraz.count(litera)**:

string = "Python is awesome, isn't it?"

substring = "is"

count = string.count(substring)

print(count)

(wynik: 2)

---można oczywiście zapisać skrótowo. Policz a w stringu:

input = "aabaa"

print (input[0].count("a"))

albo

print ("aabaa".count("a")

---w podanych stringach sprawdż który jest dłuższy a potem sprawdź ile razy w dłuższm występuje krótszy)

input = ["LJLJyYZAnPbjNLJLJ LJ", "eVj eVjeVjqSkoBAApcXaufLeVj"]

for i in input:

list=i.split()

if len(list[0])<len(list[1]):

long = list[1]

short = list[0]

else:

long = list[0]

short = list[1]

print long.count(short)

(wynik: 4 3)

---wypisz najczęśćiej występującą literę, jeśli jest więcej takich liter, to wypisz je jako listę:

input = ("grape", "apple", "blueberry")

wynik:

['g', 'r', 'a', 'p', 'e'] # każda litera wyst. 1 raz, więc mamy listę

p # p występuje 2x reszta po raz, więc wypisuje tylko p

['b', 'e', 'r'] # na liście jest każda z liter występująca po 2 razy

dla każdego wyrazu na liście input (a raczej tupli/krotce) sprawdzamy: dla każdej litery w wyrazie, czy ta litera nie jest na liście letters, jeśli jej nie ma to sprawdzamy ile razy wystąpiła i czy więcej razy niż obecne max. Jeśli częściej, to wrzucamy ją na listę poprzez to, że nadpisujemy(!), nawet jeśli tam było już więcej liter występiujących np tylko 1 raz, to się skasuje cała lista, bo teraz wiemy, że są litery występujące np 2 razy). letters = [litera] i przypisujemy też nowe max. Jeśli ilość występowania kolejnych liter jest równa obecnej ilości max to dodajemy do listy. Na koniec sprawdzamy długość listy letters i jeśli mamy tylko jedną literę na liście to wtedy drukujemy samą literę a jeśli więcej to drukujemy całą listę.

for wyraz in input:

max = 0

letters=[]

for litera in wyraz:

if litera not in letters:

if wyraz.count(litera) > max:

letters = [litera]

max = wyraz.count(litera)

elif wyraz.count(litera) == max:

letters.append(litera)

if len(letters) == 1:

print(letters[0])

else:

print(letters)

--moje:

for wyraz in input:

lista = []

max = 0

for litera in wyraz:

if max < wyraz.count(litera):

max = wyraz.count(litera)

lista = [litera]

elif max == wyraz.count(litera) :

if litera not in lista: #może być też tutaj a nie pod for i też działa

lista.append(litera)

if len(lista) == 1:

print (lista[0])

else:

print lista

----metoda **swapcase()** zamiania duże litery na małe i małe na duże, **lower()** na małe a **upper()** na duże:

text = "aBc"

print("Before: " + text)

text = text.swapcase()

print("After: " + text)

wynik:

Before: aBc

After: AbC

duze = text.upper() # wynik ABC

male = text.lower() # wynik abc

---w podanych wyrazach zamień wielkość liter i je wypisz

input = ("Ala", "kOT")

for wyraz in input:

print ( wyraz.swapcase() )

(wynik:

aLA

Kot)

---**index()** zwaraca index znaku albo ciągu znaków (pierwszej jego litery):

sentence = "PixBlocks"

print(sentence.index("P"))

print(sentence.index("Block"))

(wynik: 0 i 3)

----podaj indeks pierwszego występowania podanej litery

input = ["plum", "p"]

for letter in input[0]:

if letter == input[1]:

print input[0].index(input[1])

break

(wynik: 0)

----**find()**

textA = "abcde"

textB = "fghij"

print(textA.find('c'))

print(textB.find('c'))

textC = "abcabc"

print(textC.find('b', 2, 5)) #Find 'b' from index 2 to 5

wyniki: 2 -1 4

6-----w podanych wyrazach sprawdź czy zawierają "a" i jeśli tak to wypisz ostatni index a jeśli nie to wypisz None:

--moje:

index=0

for wyraz in input:

if wyraz.count("a") == 1:

print (wyraz.find("a"))

elif wyraz.count("a") > 1:

max = 0

for i in range (len(wyraz)):

if wyraz[i] == "a":

max = i

print max

else:

print None

--pix:

for i in input:

last = len(i) - 1 - i[::-1].find('a')

if last == len(i):

print("None")

else:

print(last)

---sprawdzenie czy tekst się zaczyna lub kończy literą lub stringiem **startswith**() i **endswith()**:

text = "abcde"

print(text.startswith("abc")) #True because first characters are "abc"

print(text.endswith("de")) #True because last characters are "de"

---sprawdź czy wyraz jest na końcu - wydrukujj Right, na początku wydrukuj Left a jeśli ani tu ani tu to wydrukuj Center

input = ["water wat", "watermelon melon", "sweet ee"]

for i in input:

list = i.split()

if list[0].startswith(list[1]):

print "Left"

elif list[0].endswith(list[1]):

print "Right"

else:

print "Center"

---pix zrobił łopatologicznie:

for i in input:

text = i.split()[0]

word = i.split()[1]

if text.startswith(word):

print("Left")

elif text.endswith(word):

print("Right")

else:

print("Center")

---wydrukuj to co jest pod indexem 3 ze stringa

sentence = "PixBlocks"

character = sentence[3]

print(character)

(wynik: B)

---wydrukuj z pierwszego elementu ciągu to co znajduje sie pod indeksem na który wskazuje drugi element ciągu

input = ["fiufju", 4]

print input[0][ input[1] ]

(wynik: j)

---policz ile razy występuje w wyrazie litera spod podanego indeksu:

input = [ "text 2", "drugi 3", "trzeci 5"]

for i in input:

lista = i.split()

wyraz = lista[0]

index = int(lista[1])

litera = wyraz[index]

print (wyraz.count(litera))

---pix skompresował maksymalnie:

for i in input:

word = i.split()[0]

index = int(i.split()[1])

print(word.count(word[index]))

--- **[ : : -1 ]**

text = "abcde"

print(text[**1:3**]) # index **1** do index **2**

wynik: bc

print(text[::]) # cały string

abcde

print(text[::-1]) # cały string odwrócony

edcba

print(text[0:-1]) # od indexu 0 bez jednego od końca << "-" usuwa ostatnie znaki

abcd

---ćwiczenie:

text = "abcdefgh"

print(text[0:-1])

print(text[0:-2])

print(text[0:-3])

wynik:

abcdefg

abcdef

abcde

---wypisz podaną ilość pierwszych liter każdego stringa:

input = [ "text 2", "drugi 3", "trzeci 5"]

for i in input:

text = i.split()[0]

ile = int(i.split()[1])

print( text[0:ile] ) # bez 0 też działa, 0 jest domyślne czyli text[:ile]

(wynik: "te" "dru" "trzec")

---podany jest wyraz, index początkowy i końcowy, wypisz literki:

input = [ "text 0 2", "drugi 3 4", "trzeci 1 5"]

for i in input:

word = i.split()[0]

start = int(i.split()[1])

end = int(i.split()[2])

print( word[start:end] )

(wynik: "te" "g" "rzec")

12---wypisz połowy wyrazów. Każdy wyraz jest parzysty:

input = ["alla", "aoieuy", "al"]

for i in input:

print (i[0:len(i)/2]) # [0:-len(i)/2] działa tak samo - odejmuje połowę.. od tyłu

(trzeba pamiętać że każde "i" to wyraz na liście i nie mogę zapisywać go jako "input[i]", bo to jest po prostu "i")

---zdanie = **"spacja lub coś innego". join(lista)**

list = ["Programming", "in", "Python", "is", "cool."]

**sentence = " ".join(list)**

print(sentence)

wynik: Programming in Python is cool.

---inne:

list = ["Programming", "in", "Python", "is", "cool."]

sentence = " \*\* ".join(list)

print(sentence)

# Programming \*\* in \*\* Python \*\* is \*\* cool.

---połącz podane wyrazy, dodając je do listy a potem wypisz za pomocą join łacząc je znakim \_

input:

blueberry

plum

watermelon

list =[]

for wyraz in input:

list.append(wyraz)

print ("\_".join(list))

wynik: blueberry\_plum\_watermelon

---pix (bo input to jest lista albo krotka(tupla). Moje rozwiązanie dodatkowo zamienia ewentualną krotkę(tuplę) w listę, ich pozostawiłoby ewentualną tuple tuplą. Ale wynik jest taki sam: string połączony\_):

list = input

print("\_".join(list))

---połącz stringi w rozsypce (oddzielone licznymi spacjami) w wyrazy:

w a t e r m e l o n

t o m a t o

p e a s

for string in input:

list = string.split()

print ("".join(list))

wynik:

watermelon

tomato

peas

---

word = "Test"

print(word)

word = list(word)

word[2] = "x"

word = "".join(word)

print(word)

Test

Text

--drugi od końca element stringa nie wiadomo czy to jeden czy dwa znaki, a ja chcę ostatni znak wypisać:

i = "apple 4 -> d"

list = i.split() # ["apple", "4", "->", "d"]

element = list[-2]

print element # wynik: -> (a więc to traktuje jak jeden element listy - bo nie ma spacji między - i > więc na liście wskoczyło jako "->")

ost = list[-1]

print ost # wynik: d

---funkcja **list(string)** zamienia stringa w listę:

plum 2 -> r

plum 1 -> x

for i in input:

lis = i.split() # ważne - lista nie może się nazywać list, bo potem funkcja się mota

wyraz = lis[0]

index = int(lis[1])

litera = lis[-1]

wyraz = list(wyraz)

wyraz[index] = litera

print("".join(wyraz))

plrm

pxum

----wypisz samogłoski (pierwszy wyraz na liście "list" i umieść je na początku w nowym wyrazie:

list = ["aoieuy", "Programming", "in", "Python", "is", "cool."]

samogloski = list[0]

for i in range (1,len(list)): # bo nie dotyczy "aoieuy"

wyraz = list[i]

samo=[]

spol=[]

for letter in wyraz:

if letter in samogloski:

samo.append(letter)

else:

spol.append(letter)

print ("".join(samo+spol))

oaiPrgrmmng

in

yoPthn

is

oocl.

-pix:

vowels = input[0].split()[1]

for i in range(1, len(input)):

word = input[i]

newWord = "" # dali od razu stringa, a potem do niego dodają litery +(plusem)

consonants = "" # znaczy spolgloski

for j in range(0, len(word)):

if word[j] in vowels:

newWord += word[j]

else:

consonants += word[j]

print(newWord + consonants)

----podane wyrazy wypisz zamianiając co druga literę na dużą. Zacznij od dużej:

for wyraz in input:

new = list(wyraz)

for j in range (0,len(new)): # od 0 do dł. listy i co drugi element (modulo 2)

if j%2 == 0:

new[j] = new[j].upper()

print ("".join(new))

--pix:

for i in input:

letters = list(i)

for j in range(0, len(letters), 2): # od 0 do dł. listy i co drugi element

letters[j] = letters[j].upper()

print("".join(letters))

----metoda **replace("znak","nowy znak")** umożliwia zamianę znaku:

sentence = "Pi\_Blocks"

print(sentence)

sentence = sentence.replace("\_", "x")

print(sentence)

Pi\_Blocks

PixBlocks

----zamień :

input = ("plum m -> o", "fig g -> r")

for i in input:

lista = i.split()

print (lista[0].replace(lista[1],lista[-1]))

pluo

fir

---wypisz wyrazy z pominięciem pierwszego znaku:

input = ["f bfeaabdfdbefgbff", "c fdfcfcdcffee"]

for i in input:

list = i.split()

print (list[1].replace(list[0],""))

beaabddbegb

fdffdffee

---metoda tekst.**isdigit()** sprawdza czy podany string jest tylko cyframi:

stringA = "0192837465"

print(stringA.isdigit())

stringB = "103ab"

print(stringB.isdigit())

True

False

--sprawdź czy podane stringi skłądają się z cyfr. Jeśli tak to zamiań je na int, a jeśli nie to wypisz False:

input = ["fwr3", 350652]

for wyraz in input:

wyraz = str(wyraz) # cyferki są int, trzeba je wpierw zmienić na str bo isdigit nie zadzała

if wyraz.isdigit():

print int(wyraz)

else:

print False

--

letter = "a"

string = "!@$"

print(**ord(letter)**) #ASCII code of 'a'

print(ord(string[1])) #ASCII code of '@'

---wypisz kod ASCII pierwszej litery każdego wyrazu w input:

for wyraz in input:

print (ord(wyraz[0]))

---dla każdego wyrazu na liście input znajdź kod ASCII dla każdej litery i wypisz wszystkie jako kody ASCII ale jako jeden string połączony spacją:

for wyraz in input:

kody=[]

for letter in wyraz:

kody.append(str(ord(letter))) # trzeba zamienić każdy kod ASCII na str, bo join musi mieć stringi

print (" ".join(kody))

---**chr(cyfra)** robi na odwrót niż ord(litera), czyli masz cyferkę to dostaniesz literę jaka jest pod nią zakodowana:

letterM = chr(77) # ord('M') = 77, więc letterM = 'M'

print(letterM)

wynik: M

letterACode = ord('A') #letterACode = 65

letterA = chr(letterACode)

print(letterA)

wynik: A

----wypisz znaki które odpowiadają podanym cyfrom:

for i in input:

print (chr(i))

---wypisz słowa ukryte w podanych ciągach:

input = ["97 112 112 108 101", "112 108 117 109"]

for i in input:

lista = i.split()

slowo =[]

for kod in lista:

slowo.append(str(chr(int(kod)))) #wpierw zmiana kodu na int, żeby chr zadziałało, a potem całość zmiana na str, żeby join zadzialało :D

print ("".join(slowo))

apple

plum

---pix od razu zrobił stringa i dodaje do niego od razu :

for i in input:

word = ""

for j in i.split():

word += str(chr(int(j)))

print(word)

---na moim też można zamienić słowo pusty string i dopisywać literki:

for i in input:

lista = i.split()

slowo =""

for kod in lista:

slowo+=str(chr(int(kod)))

print (slowo)

---z podanych ciągów znaków wypisz ten, który ma najwyższy kod ASCII:

input=["jMl2X80w", "QTXSULjpD"]

for wyraz in input:

max = 0

for litera in wyraz:

if max <= ord(litera):

max = ord(litera)

print(chr(max))

w

p

---zaszyfruj podane ciągi znaków szyfrem Cezara - każda litera jest zastąpiona literą występującą 3 po niej. Ostatnie litery są zastąpione A, B i C.

input =["LJRCLMS", WHNXVO CSMFBLZB YCELYM"]

for ciag in input:

lista = ciag.split()

newlista=[]

for wyraz in lista:

nowe=""

for letter in wyraz:

if letter == "X":

nowe+="A"

elif letter == "Y":

nowe += "B"

elif letter == "Z":

nowe +="C"

else:

nowe += chr(ord(letter)+3)

newlista.append(nowe)

print (" ".join(newlista))

OMUFOPV

ZKQAYR FVPIEOCE BFHOBP

---pix (zamienili spacje na spacje i nie musieli splitować):

for i in input:

output = ""

for j in i:

if j == " ":

output += " "

elif ord(j) <= 87:

output += chr(ord(j) + 3)

else:

output += chr(ord(j) - 23)

print(output)

---stwórz zbiór od 1 do 10 i go wypisz:

list=[]

for i in range(1,11):

list.append(i)

print (set(list)) # wynik: set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

---inaczej:

list = [i for i in range (1,11)]

set = set(list)

print set

---albo od razu:

set = set( [i for i in range (1,11)])

print set

---pix:

set = set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

print(set)

----jeśli lista zawiera powtarzające się znaki to **set(lista)** je usunie:

list = [1, 2, 3, 1, 2, 3]

set = set(list)

print("List: " + str(list))

print("Set: " + str(set))

List: [1, 2, 3, 1, 2, 3]

Set: set([1, 2, 3])

---utwórz zmienną typu set i dodaj do niej liczby z ciągów liczbowych za pomocą metody **add()** a potem wypisz wszystkie te liczby. Uwaga: poprzez add można dodawać każdą liczbę tylko pojedyńczo.

88 62 51

18 53 66 82

76 14 25 78 97

16 95 84 22 88

80 92 51 54 54

57 63 72 67

71 12 83 53 29

92 58 58 61

50 22 26 17

set = set()

for i in input:

x = i.split()

for j in x:

set.add(int(j)) # int, bo inaczej to wyszła lista ze stringami set(['63','67'...])

print set # gdyby print był pod "for j in x" to by drukował sety po kolei np. set([88,62,51])

set([63, 66, 67, 71, 72, 12, 76, 14, 78, 16, 80, 17, 18, 82, 83, 84, 22, 88, 25, 26, 92, 29, 95, 97, 50, 51, 53, 54, 57, 58, 61, 62])

---stwórz listę z podanych liczb i ją wypisz a potem stwórz zbiór i go wypisz:

3

7

3

10

list=[]

for i in input:

list.append(i)

secik = set(list) # może być set zamiast secik jeśli będzie równo z printami

print list

print secik

[3, 7, 3, 10]

set([3, 7, 10]) # set układa liczby po kolei rosnąco

albo:

print input

print set(input)

---dodajemy add a usuwamy **remove()**

set = set()

set.add(1)

set.add(2)

set.add(3)

print(set)

set([1, 2, 3])

set.remove(2)

print(set)

set([1, 3])

----można też usuwać metodą **discard()**, różni się od remove() tym, że jak nie ma elementu, który trzeba usunąć to discard() nie wywala błedu:

set = set([2, 4, 6, 8, 10])

print("Before remove: " + str(set))

Before remove: set([8, 2, 10, 4, 6])

set.remove(2);

print("After remove: " + str(set))

After remove: set([8, 10, 4, 6])

set.discard(2) #Nothing happened

#set.remove(2) error

---z podanych liczb stwórz zbiór a następnie usuń liczby 1 i 3 jeśli są i go wypisz:

set = set()

for i in input:

set.add(i)

set.discard(1)

set.discard(3)

print (set)

---teraz zrób to samo ale usuń liczby od 1 do 3 JEŚLI występują. użyj metody remove():

set = set()

for i in input:

set.add(i)

for j in range (1,4): # lepiej zrobić jak pix poniżej

if j in set:

set.remove(j)

print set

---pix zaszalał z pętlami. (Może być przydantne jeśli trzeba usunąć większy przedział ze zbioru):

set = set()

for i in input:

set.add(i)

for i in range(1, 4):

if i in set:

set.remove(i)

print(set)

---metoda **pop()** usuwa pierwszy element ze zbioru i go zwraca:

set = set([1, 2, 3, 4, 5])

print("Before: " + str(set))

print(set.pop())

print("After: " + str(set))

Before: set([1, 2, 3, 4, 5])

1

After: set([2, 3, 4, 5])

---dodaj podane liczby do zbioru a potem wypisz 3 pierwsze usuwając je ze zbioru.

set = set()

for i in input:

x = i.split()

for j in x:

set.add(j) # wypisze set(['58', '16', '49', '38', '28', '44', '51', '20', '61', '42', '24', '62'])

set.add(**int**(j)) # wypisze ułożony set([42, 44, 24, 49, 28, 51, 58, 16, 38, 61, 20, 62])

print set

for a in range (3):

print (set.pop())

--wersja rozbudowana:

lista = ['58', '16', '49', '38', '28', '44', '51', '20', '61', '42', '24', '62']

set1 = set(lista)

print set1

for a in range (3):

print (set1.pop())

# nie zrobiliśmy int więc lista się nie posortowała i inne pierwsze 3 elementy zostały wywalone za set1 niż z set

set = set()

for j in input:

set.add(int(j))

print set

for a in range (3):

print (set.pop())

print set

zbior = set()

for i in input:

list = i.split()

for j in list:

zbior.add(int(j))

for i in range(3):

print zbior.pop()

set(['58', '16', '49', '38', '28', '44', '51', '20', '61', '42', '24', '62']) # set1

58

16

49

set([42, 44, 24, 49, 28, 51, 58, 16, 38, 61, 20, 62]) # set

42

44

24

set([49, 28, 51, 58, 16, 38, 61, 20, 62]) # set po trzykrotnym zastosowaniu pop() stracił 3 pierwsze elementy. (set1 też je stracił, tylko nie wydrukowałam tego.)

--pix:

set = set()

for i in input:

for j in i.split():

set.add(int(j))

print(set.pop())

print(set.pop())

print(set.pop())

----metoda **union()** liczy sumę zbiorów:

setA = set([1, 2])

setB = set([3, 4])

print("Set A: " + str(setA)) # Set A: set([1, 2])

print("Set B: " + str(setB)) # Set B: set([3, 4])

union = setA.union(setB)

print("Union of set A and B: " + str(union)) # Union of set A and B: set([1, 2, 3, 4])

---liczby w dwcóch kolumnach, te z lewej dodaj do jednego zbioru a te z prawej do drugiego, na koniec połącz oba zbiory metodą **union():**

'2 10'

'5 10'

'2 10'

setA = set()

setB = set()

for i in input:

setA.add(int(i.split()[0]))

setB.add(int(i.split()[1]))

print (setA.union(setB))

set([2,5,10])

---metoda **intersection()** oblicza iloczyn zbiorów (część współną):

setA = set([1, 2, 3])

setB = set([2, 3, 4])

print("Set A: " + str(setA))

print("Set B: " + str(setB))

intersection = setA.intersection(setB)

print("Intersection of set A and B: " + str(intersection))

Set A: set([1, 2, 3])

Set B: set([2, 3, 4])

Intersection of set A and B: set([2, 3])

---stwórz 3 zbiory z każdej kolumny a następnie znajdź ich część współną za pomocą metody **intersection():**

14 19 8

12 14 14

2 23 19

19 1 20

setA, setB, setC = set(), set(), set()

for i in input:

setA.add(i.split()[0])

setB.add(i.split()[1])

setC.add(i.split()[2])

print setA.intersection(setB.intersection(setC)) # może być setA.intersection(setB,setC)

set(['14', '19'])

pix:

a, b, c = set(), set(), set()

for i in input:

a.add(i.split()[0])

b.add(i.split()[1])

c.add(i.split()[2])

print(a.intersection(b, c))

---metoda **intersection\_update()** usuwa elemety ze zbioru, które nie są jednocześnie obecne w obecnym zbiorze oraz zbiorze podanym jako argument (sama metoda **intersection()** nie zmienia zbiorów tylko tworzy nowy - część współną, natomiast **intersection\_update()** usuwa elementy ze zbioru na którym jest zastosowana:

setA = set([1, 2, 3, 4, 5])

setB = set([3, 4, 5, 6, 7])

print("Set A: " + str(setA))

print("Set B: " + str(setB))

**setA**.**intersection\_update(**setB**)**  # setA zostanie zmieniony, setB bez zmian

print("Set A - after: " + str(setA))

print("Set B - after: " + str(setB))

Set A: set([1, 2, 3, 4, 5])

Set B: set([7, 3, 4, 5, 6])

Set A - after: set([3, 4, 5])

Set B - after: set([7, 3, 4, 5, 6])

---metoda **difference()** służy do obliczania różnicy zbiorów:

setA = set([1, 2, 3])

setB = set([2, 3, 4])

aDifferenceB = setA.difference(setB) # zbiór A minus zbiór B

bDifferenceA = setB.difference(setA) # zbiór B minus zbiór A

print(aDifferenceB)

print(bDifferenceA)

set([1])

set([4])

---metoda **symmetric\_difference()** liczy różnicę w obu zbiorach i sumuje ją razem. Dla powyższych zbiorów wynik byłby set([1,4])

setA = set(["Tom", "Jerry", "Eric"])

setB = set(["Eric", "Bill", "Tom"])

symmetricDifference = setA.symmetric\_difference(setB)

print(symmetricDifference)

set(['Bill', 'Jerry'])

---dodaj do dwóch zbiorów a potem wylicz róznicę zbior A minus B i zbioru B minuz zbiór A:

9 10

10 2

3 8

6 1

4 10

2 5

a,b = set(), set()

for i in input:

a.add(i.split()[0])

b.add(i.split()[1])

print a.difference(b)

print b.difference(a)

set(['4', '6', '9', '3'])

set(['1', '8', '5'])

---podane osoby brały udział w 3ch wydarzeniach. Podaj osoby, które brały udział tylko w jednym. Wykorzystaj metodę symmetric\_difference():

Vicky Hannah Hubert

David Simon Alice

Simon Matt Matt

nie da się zrobić symmetric\_difference() z trzech zbiorów.

--pix zrobił z dwóch i zadowoleni, bo nawet nie zauważyli byka.

setA, setB, setC = set(), set(), set()

for i in input:

a, b, c = i.split()[0], i.split()[1], i.split()[2]

setA.add(a)

setB.add(b)

setC.add(c)

symmetricDifference = setA.symmetric\_difference(setB)

print(symmetricDifference)

----metoda **update()** dodaje do zbioru więcej niż jeden element:

set = set([1, 2])

print("Before update: " + str(set))

set.update([3, 4, 5])

print("After update: " + str(set))

Before update: set([1, 2])

After update: set([1, 2, 3, 4, 5])

---doda podane liczby do zbioru za pomocą update a potem wypisz liczby w zbiorze po kolei

15 14 2

14 13 2

set = set()

for i in input:

set.update(i.split())

for i in set:

print(i)

15

14

----metoda **isdisjoint()** sprawdza czy są elementy wspólne, zwraca True albo False:

setA = set([1, 2, 3])

setB = set([4, 5, 6])

setC = set([2, 3, 4])

print(setA.isdisjoint(setB)) #None joint elements

print(setA.isdisjoint(setC)) #Two joint elements: 2, 3

print(setB.isdisjoint(setC)) #One joint element: 4

---trzy zestawy liczb, dodaj każdy do osobnego setu a potem sprawdź czy A i B sa rozłączne, potem B i C oraz A i C metodą isdisjoint():

input = ["7 11 6 2 3","6 10 18", "2 16 4"]

a,b,c=set(),set(),set()

a.update(input[0].split())

b.update(input[1].split())

c.update(input[2].split())

print (a.isdisjoint(b))

print (b.isdisjoint(c))

print (a.isdisjoint(c))

False True False

--- metoda **issuperset()** sprawdza czy dany zbiór jest nadzbiorem drugiego zbioru:

setA = set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) # superset of set B

setB = set([1, 2, 3, 4, 5]) # Subset of set A

setC = set([1, 2, 3, 4, 5, 100]) # It set an element that's not in setA - 100

print(setA.issuperset(setB)) # True

print(setA.issuperset(setC)) # False

---pierwszy zestaw dodaj do zbioru x a następnie sprawdź czy jest nadzbiorem pozostałych:

input = [ "1 4 9 14 16 19 20", "2 9 10 6", "4 20 1 9"]

x = set()

x.update(input[0].split())

for i in range (1,len(input)):

ciag = input[i]

print x.issuperset(ciag.split())

False True

---metoda **issubset()** sprawdza czy dany zbiór jest podzbiorem drugiego:

setA = set([1, 2, 3])

setB = set([1, 2, 3, 4, 5])

print(setA.issubset(setB)) # True

---na wejściu podano ciągi liczbowe zawierające znak |. Sprawdź metodą issubset() czy ciąg po lewej stronie tego znaku jest podciągiem tego po prawej.

input = ["2 | 2 4 5 6", "1 2 | 1 5 6"]

for i in input:

dwaciagi = i.split("|")

malyset = set(dwaciagi[0].split())

duzyset = set (dwaciagi[1].split())

print (malyset.issubset(duzyset))

# True True

--pix, namotali nawet nie chce mi sie w to wgłębiać:

for i in input:

left, right = set(), set()

elements = i.split()

for i in elements:

if i != '|':

left.add(int(i))

else:

break

for i in reversed(elements):

if i != '|':

right.add(int(i))

else:

break

print(left.issubset(right))

---**frozenset** to taki set, którego nie da się edytować. Niezmianialny. Np. nie działa metoda add.

frozenSet = frozenset([1, 2, 3])

print(frozenSet)

#frozenSet.add(1)

#error: 'frozenset' object has no attribute 'add'

#Frozen set is immutable

dictionary = { "Name" : "Elizabeth", "Phone number" : 123456789 }

dictionaryKeys = frozenset(dictionary)

print(dictionaryKeys)

# frozenset(['Phone number', 'Name'])

--- LISTY SKŁADANE

---można zapodać coś dla każdego elelmentu w liście, np. zmianić floaty w integery albo w stringi. Element może się nazwac tez np x, albo inaczej.

list = [1.2, 3.1456789, 22.7, 28.9, 50.1]

print(list) # [1.2, 3.1456789, 22.7, 28.9, 50.1]

newList = [int(element) for element in list]

print(newList) # [1, 3, 22, 28, 50]

newList = [str(x) for x in list]

print(newList) # ['1.2', '3.1456789', '22.7', '28.9', '50.1']

---można też dodać instrukcję warunkową (w tym przypadku na nowej liście pojawią się tylko **parzyste**)

list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(list) # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

newList = [element for element in list if element % 2 == 0]

print(newList) # [2, 4, 6, 8, 10]

---można też stworzyć nową listę, nie trzeba modyfikować istniejącej:

list = [i for i in range(1, 11)]

print(list) # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

--można też od razu w nowo utwarzanej liście zastosować warunek, np, żeby wypisać tylko **nieparzyste** z podanego zakresu:

list = [i for i in range(1, 11) if i%2 !=0]

print(list)

---stwórz nową listę zawierającą **długości** podanych słów:

input[0] = ("blackberry strawberry banana strawberry blueberry coconut ")

list = input[0].split()

newList = [len(element) for element in list]

print(newList) # [10, 10, 6, 10, 9, 7]

---wypisz tylko nieparzyste:

input = ["20 17 39 30 16 "]

list = input[0].split()

oddNo = [int(element) for element in list if int(element) % 2 != 0]

print (oddNo) # [17, 39]

---wykorzystując listę składaną i funkcję wypisz liczby pierwsze:

moje:

input = ["2 90 1 16 87 6 31 71"]

def prime(x):

if x<2: # bez tego modułu też działa

return False

for i in range(2,x+1):

if x%i == 0:

if x == i:

return True

else:

return False

list = input[0].split()

primeList = [int(element) for element in list if prime(int(element))]

print (primeList) # [2, 31, 71]

---jeżeli zniknie int w liście to będzie lista taka sama ale ze stringów ['2', '31', '71']. Poniższa wersja nie ma już pierwszego if-a a sprawdzanie pierwszych też działa:

def prime(x):

for i in range(2,x+1):

if x%i == 0:

if x == i:

return True

else:

return False

list = input[0].split()

primeList = [element for element in list if prime(int(element))]

print (primeList) # ['2', '31', '71']

---wypisz liczby z podanych zakresów, użyj jednej pętli for i listy składanej:

input = ['3 6', '1 3']

for j in input:

new = [i for i in range (int(j.split()[0]), int(j.split()[1])+1)]

print new

[3, 4, 5, 6]

[1, 2, 3]

-----**any**

list = [1, 3, 4, 5, 7]

print(any(i % 2 == 0 for i in list)) # True

----spr. czy występuje chociaż jedna dodatnia liczba w podanych ciągach liczb:

12 -54 -13 -62 -20 -23 -57

-64 -40 -28 -6 -26 -14

for i in input:

list = i.split()

print (any(int(element)>0 for element in list))

True

False

---**all**

listA = [1, 3, 5, 7, 9]

listB = [2, 3, 4, 5, 6]

print(all(i % 2 == 1 for i in listA)) # True

print(all(i % 2 == 1 for i in listB)) # False

---sprawdź czy wszystkie liczby w podanych ciągach są niepodzielne przez 3:

38 49 65 52

73 98 81 17 64

for i in input:

list = i.split()

print(all(int(x) % 3 !=0 for x in list))

True

False

--pix jeszcze bardziej skompresował:

for i in input:

print(all(int(x) % 3 != 0 for x in i.split()))

---funkcja **map** przekonwertowuje wszystkie elememty listy:

list = ["1", "2", "3", "4", "5"]

list = map(int, list) # (mapuj int na listę)

print(list) # [1, 2, 3, 4, 5]

---za pomocą funkcji map wypisz listy liczb podniesionych do sześcianu:

def toCube(number):

return int(number) \*\* 3;

for i in input:

print(map(toCube, i.split()))

---wypisz ciągi liczb z ciągu fibinacciego, podanne liczby to indeksy tych liczb:

def fibonacci(n):

if (n < 3):

return 1

else:

return fibonacci(n - 1) + fibonacci(n - 2)

for i in input:

numbers = map(fibonacci, map(int, i.split())) # zwróć uwagę, brak nawiasów i argumentu

print(numbers)

---ćwiczenie: zaokrąglij elementy z listy input za pomocą round i map:

input = ["6845.95841 7725.76331 5475.02441", "......... ....."]

for i in input:

list = i.split()

print list # ['6845.95841', '7725.76331', '5475.02441']

list1 = map(float, list)

print list1 # [6845.95841, 7725.76331, 5475.02441]

list2 = [float(element) for element in list] # zrobi to samo co ten wyżej

print list2 # [6845.95841, 7725.76331, 5475.02441]

list3 = map(round,list1)

print list3 # [6846.0, 7726.0, 5475.0]

list4 = map(int,list3)

print list4 # [6846, 7726, 5475]

--w skrócie, kiedy się wklei wszystko w jedną linijkę:

for i in input:

list4 = map(int,map(round,map(float, i.split())))

print (list4)

---**filter**

def "rom():

if word == word [::-1]:

return True

return False

list = ["rotor", "Python", "radar", "rabbit", "car", "level", "madam"]

print(filter(isPalindrome, list)) # ['rotor', 'radar', 'level', 'madam']

PS: print(**map**(isPalindrome, list)) # [True, False, True, False, False, True, True]

---z podanej listy liczb odfiltruj liczby pierwsze:

def prime(x):

if x<2:

return False

for i in range(2,x+1):

if x%i == 0:

if x == i:

return True

else:

return False

for i in input:

list = i.split()

list\_int = map(int, list)

list\_prime = filter(prime, list\_int)

print map(str,list\_prime)

lub w skrócie:

for i in input:

print (map(str,filter(prime, map(int, i.split()))))

--jeśli przerzucimy zamianę liczby w integer do funkcji:

def prime(x):

if int(x)<2:

return False

for i in range(2,int(x)+1):

if int(x)%i == 0:

if int(x) == i:

return True

else:

return False

for i in input:

print ( filter(prime, i.split()) )

--pix zrobił podobnie ale jakieś dziwactwo w funkcji, w sensie pierwiastek:

def isPrime(number):

if int(number) < 2:

return False

for i in range(2, int(**int(number) \*\* 0.5**) + 1):

if int(number) % i == 0:

return False

return True

for i in input:

print(filter(isPrime i.split()))

--Troche teorii o pierwiastkowaniu:

There are at least 3 ways to do a square root in Python:

- math.sqrt,

- the '\*\*' operator

- pow(x,.5).

pow and \*\* are equivalent; math.sqrt doesn't work for complex numbers, and links to the C sqrt() function

---odnajdź palindromy w podanych przykładach i wypisz ich dugość w postaci listy. Wykorzystaj "filter" i "map":

input=["hone city eye smart city", "love refer stats human refer"]

--miało być map, ale zrobiłam z list.append():

for i in input:

list = i.split()

list\_len = []

for word in list:

if word == word[::-1]:

list\_len.append(len(word))

print list\_len

--wersja z listą składaną:

for i in input:

list = i.split()

new = [len(word) for word in list if word == word[::-1]]

print new

--wersja z map i filter:

def palindrom(word):

if word == word[::-1]:

return True # else return False jest zbędne, bez tego też działa

for i in input:

palindromy = filter(palindrom,i.split())

print map(len,palindromy)

---pix zrobił tak samo, tylko jeszcze bardziej skompresował ostatnią linijkę i nie usunął else:

def isPalindrome(word):

if word == word[::-1]:

return True

else:

return False

for i in input:

print(map(len, filter(isPalindrome, i.split())))

---z podanych ciągów liczb wypisz najpierw parzyste a potem nieparzyste pod postacią nowych list, używając map i filter:

95 35 3 8 43

32 63 34 44

def even(x):

if x%2 ==0:

return True

def odd(x):

if x%2 !=0:

return True

for i in input:

lista = map(int,i.split())

parzyste = filter(even,lista)

nieparzyste = filter (odd,lista)

print (parzyste + nieparzyste)

[8, 95, 35, 3, 43]

[32, 34, 44, 63]

--pix jeszcze ciekawiej to zrobił:

def isEven(n):

return n % 2 == 0

def isOdd(n):

return n % 2 == 1

for i in input:

numbers = map(int, i.split())

even = filter(isEven, numbers)

odd = filter(isOdd, numbers)

even.extend(odd) # funkcja **extend**, która najwodoczniej przedłuża jedną listę o drugą

print(even)

---funkcja **zip**:

listA = [1, 2, 3, 4] #longer list, 4 not zipped

listB = ["one", "two", "three"]

list = zip(listA, listB)

print(list) # [(1, 'one'), (2, 'two'), (3, 'three')]

----połącz podane liczby w pary na liście:

58 80 # mają wystąpić na liście jako krotka (tupla): (58, 80)

87 77

95 18

listA, listB = [], []

for i in input:

listA.append(int(i.split()[0])) # listA=[58, 87, 95]

listB.append(int(i.split()[1])) # listB=[80, 77, 18]

list = zip(listA, listB)

print(list) # [(58, 80), (87, 77), (95, 18)]

--moje rozpisane:

lewa,prawa = [],[]

for para in input:

list = map(int,para.split())

lewa.append(list[0])

prawa.append(list[1])

print zip(lewa,prawa)

---wykorzystująć zip i map stwórz listę par (dana liczba, jej odwrotna liczba). Wypisz elementy w postaci zmiennooprzecinkowej float:

(liczba odwrotna y to taka, która pomnożna z daną liczbą x daje 1 => (xy = 1) czyli y =1/x)

input = [6,2,8]

list = map(float,input)

odwr = [1/float(element) for element in list]

print zip(list,odwr) # [(6.0, 0.16666666666666666), (2.0, 0.5), (8.0, 0.125]

---skrótowo i przejrzyście:

liczba=map(float,input)

odwrotna = [1/i for i in liczba]

print zip(liczba,odwrotna)

---pix zaszaleli:

def inverseNumber(n):

return math.pow(n, -1)

print(zip(map(float, input), map(inverseNumber, map(int, input))))

---mamy dane listy wyrazów, zrób listę par (wyraz, True), jeśli palindrom, a jeśli nie, to (wyraz, False):

rabbit rabbit rotor

radar level

def palindrom(word):

if word ==word[::-1]:

return True

else:

return False

for i in input:

list = map(palindrom, i.split())

print zip(i.split(),list)

[('rabbit', False), ('rabbit', False), ('rotor', True)]

[('radar', True), ('level', True)]

---stwórz listę która zawiera tylko liczby pierwsze z podanych. (odfiltruj) a potem listę która zawiera sumy cyfr, np jesli liczba pierwsza = 81 to suma to 8+1 = 9, a na koniec zrób liste i połacz te liczby w pary, czyli np. [(81,9)]

def prime(x):

for i in range(2, x+1):

if x%i == 0:

if x ==i:

return True

else:

return False

primes = filter(prime,input)

sumlist = []

for i in primes:

right = i%10

left = i // 10

sumlist.append(left+right)

print zip(primes,sumlist)

--nie wiedzieć czemu nie działa, dopiero po podłożeniu pixblocksowej definicji funkcji "prime" działa... - ale wydaje mi się że sa błedy w ich rozwiązaniu, sorry ale 75 nie jest liczbą pierwszą:

input = [78,75,50,89,26,43,42,75,60,42,36,94,37,62,55,27,94,84,57,43]

def prime(n):

for i in range(2, int(n \*\* 0.5) + 1):

if n % i == 0:

return False

return True

primes = filter(prime,input)

sumlist = []

for i in primes:

right = i%10

left = i // 10

sumlist.append(left+right)

print zip(primes,sumlist)

---pix ma tez ciekawą funkcję sumującą - najpierw zamienia cyfry na string a potem każdą z nich iteruje w pętli, zamienia spowrotem w int i sumuje.

def isPrime(n):

for i in range(2, int(n \*\* 0.5) + 1):

if n % i == 0:

return False

return True

def strSum(string):

sum = 0

for i in string:

sum += int(i)

return sum

primeNumbers = filter(isPrime, input)

digitsSum = map(strSum, map(str, primeNumbers))

print(zip(primeNumbers, digitsSum))

---ładniejsza wersja funkcji prime:

def prime(n):

for i in range(2, n):

if n % i == 0:

return False

return True

------------------------REKURENCJA, czyli używanie funkcji w samej sobie---------------------

---obliczaie sumy n-kolejnych liczb naturalnych

def sum(number):

if (number < 1):

return 0

else:

return number + sum(number - 1)

for i in input:

print(sum(i))

---moje:

input = [0,1,2,3,4]

def sum(n):

if n ==0:

return 0

else:

return n + sum(n-1)

for i in input:

print (i,sum(i))

(0, 0)

(1, 1)

(2, 3)

(3, 6)

(4, 10)

----silnia:

input = [0,1,2,3,4,5,6]

def silnia(n):

if n<=1:

return 1

else:

return n \* silnia(n-1)

for i in input:

print (i, silnia (i))

(0, 1)

(1, 1)

(2, 2)

(3, 6)

(4, 24)

(5, 120)

(6, 720)

silnia pix:

def factorial(number):

factorialTemp = 1

if number in [0, 1]:

return 1

else:

for i in range(2, number + 1):

factorialTemp = factorialTemp \* i

return factorialTemp

for i in range(10):

print(str(i) + "! = " + str(factorial(i)))

---fibbonaci:

input = [0,1,2,3,4,5,6]

def fib(n):

if abs(n) <2: # bez abs - wat. bezwzględnej, program się zawesi - bo nie wie co robić przy n=0, a raczej to po else: niekończąca się pętla

return 0

elif n == 2:

return 1

else:

return (n-1) + fib(n-2)

for i in input:

print (i, fib(i))

(0, 1)

(1, 1)

(2, 1)

(3, 2)

(4, 3)

(5, 5)

(6, 8)

----funkcja licząca potęgi liczby 2:

def power(p):

if p == 0:

return 1

else:

return 2 \* power(p - 1)

for i in range(10):

print(power(i))

1

2

4

8

16

32

64

128

256

512

----to samo ale bez rekurencji:

def power(n):

if n == 0:

return 1

else:

wynik = 2

for i in range (n-1):

wynik \*= 2

return wynik

----podane pary liczb: podstawa i wykładnik potęgi, wykonaj potęgowanie:

2 8

1 1

2 2

def potega(x,y):

if y == 0:

return 1

else:

wynik = x

for i in range (y-1):

wynik\*=x

return wynik

for j in input:

x=int(j.split()[0])

y=int(j.split()[1])

print (potega(x,y))

256

1

4

---gdy funkcja jest rekurencyjna:

def potega(x,y):

if y == 0:

return 1

else:

return (x \* potega(x, y-1))

for j in input:

x=int(j.split()[0])

y=int(j.split()[1])

print (potega(x,y))

---pix zrobił podobnie, tylko wyrzucił integery do definicji funkcji:

def pow(n, i):

if int(i) == 0:

return 1

else:

return int(n) \* pow(int(n), int(i) - 1)

for i in input:

print(pow(i.split()[0], i.split()[1]))

---fibbonaci iteracyjnie:

def fibonacci(n):

a, b = 0, 1

if n == 0:

return a

elif n == 1:

return b

else:

for i in range(1,n): # a nie 2 do n? range(2,n) bo n==1 jest już powyżej

c = a + b

a = b

b = c

return b

for i in range(10):

print(fibonacci(i))

--- inaczej

def fib(n):

pierwszy = 0

drugi = 1

if n == 0:

return pierwszy

elif n==1:

return drugi

else:

for i in range (2,n):

trzeci = pierwszy + drugi

pierwszy = drugi

drugi = trzeci

return drugi

---czy pierwsza (prime):

def czy\_pierwsza(n):

for i in range(2, n):

if n % i == 0:

return False

return True

---"odwrotność liczby" tutaj oznacza liczbę pisaną od tyłu do przodu czyli string[::-1]

input = [1234,1221, 3443, 3452]

--pix:

def makePalindrome(n, t = 0):

if t == 10:

print("False")

elif str(n) == str(n)[::-1]:

print(n)

else:

makePalindrome(n + int(str(n)[::-1]), t + 1)

for i in input:

makePalindrome(i)

---znajdź najmniejszą wspólną wielokrotność NWW liczb (do skumania):

2 75

24 73

def gcd(a, b):

if (a == b):

return a

elif(a>b):

return gcd(a-b,b)

else:

return gcd(a,b-a)

def lcm(x, y):

return x \* y / gcd(x, y)

for i in input:

numbers = i.split()

print(lcm(int(numbers[0]),int(numbers[1])))

1980

140

--------------------------------------GENENRATORY-------------------------------------------------------

def generator():

yield "Generator"

print(next(generator()))

---przykład:

def digits(): # generator "digits()"

yield 0

yield 1

yield 2

yield 3

yield 4

yield 5

yield 6

yield 7

yield 8

yield 9

generator = digits() # przypisujemy generator do zmiennej "generator"

print(next(generator))

print(next(generator))

print(next(generator))

----wypisz kolejne potęgi liczby 2 tyle razy ile podano w input:

moje:

def powersOfTwo():

for i in range(1,6):

yield 2\*\*i

x = powersOfTwo() # trzeba przypisać do zmiennej. Bez tego trzebaby wypisać nazwę funkcji generatora z nawiasami i wtedy wypisze tylko pierwszy yield tyle razy co w input (print(next(powersOfTwo() ))

for i in range (input[0]):

print(next(x))

--pix:

input = [4]

def powersOfTwo():

yield 2

yield 2\*2

yield 2\*\*3

yield 2\*\*4

yield 2\*\*5

generator = powersOfTwo()

for i in range (input[0]): # nie może być sam "input", bo to jest lista

print (next(generator))

2

4

8

16

----7 liczb pierwszych wypisz tyle ile podane:

input = [2]

def primes():

yield 2

yield 3

yield 5

yield 7

yield 11

yield 13

yield 17

generator = primes()

for i in range (input[0]):

print (next(generator))

2

3

---generator przyjmuje też argument jak w funkcji. Wartość n jest zapamiętywana do użycia w następnym generatorze:

def addTwo(number):

n = number + 2

yield n # n= 5

n += 2 # n=5 jest zapamiętane, żeby użyć w następnm yield

yield n

generator = addTwo(5)

print(next(generator))

print(next(generator))

7

9

----

input =['zq', 2]

def gen(ciag):

ciag = ciag\*2

yield ciag

ciag\*=2

yield ciag

ciag\*=2

yield ciag

generator = gen(input[0])

for i in range(input[1]):

print (next(generator))

zqzq

zqzqzqzq

---stwórz generator który odwróci podane słowo i wypisze jego litery tyle razy ile podano:

watermelon

6

def gen(word):

word = word[::-1]

for i in word:

yield i

generator = gen(input[0])

for i in range(input[1]):

print next(generator)

n

o

l

e

m

r

---pix:

def reverseWord(word):

for i in range(len(word) - 1, -1, -1): #nie kumam tych "-1"-nek..

yield word[i]

generator = reverseWord(input[0])

for i in range(input[1]):

print(next(generator))

---generator może być także nieskończony, wystarczy wykorzystać pętlę while:

def numbers():

n = 1

while True:

yield n

n += 1

generator = numbers()

for i in range(9):

print(next(generator)) # (1,2,3,4,5,6,7,8,9)

----generator, który stworzy kolejne potęgi liczby:

input=[2,3]

def powers(number):

e = 1

while True:

yield number \*\* e

e += 1

generator = powers(input[0])

for i in range(input[1]):

print(next(generator)) # 2,4,8

----moje:

def potega(n):

a = n

while True:

yield n

n = n\*a # tu nie może być \*n, bo n się zmienia. "a" przechowuje n niezmieznione

generator = potega(input[0])

for i in range (input[1]):

print (next(generator))

----zrób generator, który wypisze kolejne liczby ciągu fibbonaciego:

input = [6]

def fib():

a = 0

yield a

b = 1

yield b

while True:

c = a+b

yield c

a = b

b = c

generator = fib()

for i in range(input[0]):

print(next(generator)) # 0 1 1 2 3 5

---pix:

def fibonacci():

a, b = 0, 1

while True:

yield a

a, b = b, a + b # to działa tylko dlatego, że jest w jednej linijce!!!

generator = fibonacci()

for i in range(input[0]):

print(next(generator))

---zrób generator, który wygeneruje kolejne wyrazy ciagu arytmetycznego - pierwszy w input to 1szy wyraz ciagu, drugi to 2gi wyraz a trzeci to ilość wyrazów ciągu do wygenerowania:

input = [83,83,11]

def ciag(pierwszy,drugi):

roznica = drugi - pierwszy

yield pierwszy

while True:

yield drugi

drugi += roznica

(równie dobrze a nawet lepiej:

roznica = drugi - pierwszy

while True:

yield pierwszy

pierwszy += roznica

generator = ciag(input[0],input[1])

for i in range (input[2]):

print next(generator) # 83,83...(11x)...83

--pix wywalił liczenie roznicy na zewnątrz:

def arithmetic(start, r):

n = start

while True:

yield n

n += r

# (zrobiłam jak niżej i też działa)

while True:

yield start

start += r

generator = arithmetic(input[0], input[1] - input[0])

for \_ in range(input[2]): # dolne podkreślenie \_ zamiast "i" kiedy nie używamy "i" w formule

print(next(generator))

---zatrzymywanie iteracji w generatorze:

def digits():

for i in range(10):

yield i

raise(StopIteration) # (bez tego też jest error więc nie wiem po co to dopisywać)

generator = digits()

for \_ in range(9):

print(next(generator))

print(next(generator)) #still OK

print(next(generator)) #error

----iteracja for po generatorze: for j in digits()

def digits():

for i in range(10):

yield i

raise(StopIteration)

while True:

yield "Without StopIteration exception."

for j in digits():

print(j) # 0 1 2....9

----jeśli usuniemy raise(StopIteration) to po wypisaniu cyfr 0 do 9 będzie w nieskończoność wypisywać "Without StopIteration exception."

--jeżeli zmienię na for j in range (12):

def digits():

for i in range(10):

yield i

raise(StopIteration)

while True:

yield "Without StopIteration exception."

generator = digits()

for j in range (12):

print(next (generator))

--wypisze cyfry 0 do 9 i wywali błąd. Jeśli usuniemy raise(StopIteration) wypisze jeszcze 2x Without StopIteration...:

def digits():

for i in range(10):

yield i

while True:

yield "Without StopIteration exception."

generator = digits()

for j in range (12):

print(next (generator))

1...

8

9

Without StopIteration exception.

Without StopIteration exception.

----stwórz generator, który będzie drukował kolejne potęgi liczby 2, a astępnie iteruj po generatorze dopóki wypisywane liczby będą mniejsze niż podana:

input=[2052733]

def potega():

a = 1

while True:

n=2\*\*a

yield n

a+=1

for i in potega():

if i<input[0]:

print(i)

else:

break # ważne. Bez tego break wypisze tyle co trzeba, ale się nie zatrzyma

2

4

8

16....

...1048576

---pix wrzucił zamiast break, "raise(StopIteration)" do generatora, co zapewne jest poprawniejsze:

def powersOfTwo(max):

n = 2

while True:

yield n

n \*= 2

if n > max:

raise(StopIteration)

for i in powersOfTwo(input[0]):

print(i)

--moje poprawione z przerzuconym raise(StopIteration) do generatora:

def potega(max):

a = 1

while True:

n=2\*\*a

if n>max:

raise(StopIteration) # musi sprawdzić zanim wypisze

yield n

a+=1

for i in potega(input[0]):

print(i)

---wypisuj liczby pierwsze generatorem dopóki będą mniejsze niż podana:

input = []

def primes(max):

for i in range(2, max):

isPrime = True

for j in range(2, i):

if i % j == 0:

isPrime = False

break

if isPrime:

yield i

for i in primes(input[0]):

print(i)

---generator z okrągłymi nawiasami jak lista składana:

digits = (i for i in range(1, 21))

for i in digits:

print(i) # 1 2 3.... 20

---stwórz generator, który będzie wypisywać długości podanych wyrazów:

input = ["gooseberry", "mandarin", "plum"]

generator = (len(i) for i in input)

for i in generator:

print(i) # 10 8 4

---stwórz generator, który wypisze liczby w podanym ciągu/stringu do trzeciej potęgi:

input = ["6 8 8 7 8"]

list = map(int,input[0].split())

generator = (i\*\*3 for i in list)

for i in generator:

print i # 216 512 512 343 512

---- systemy liczbowe:

------napisz program, który zamieni podane liczby na system jedynkowy:

input = ["3", "5"]

for i in input:

jedynki = ""

for \_ in range (int(i)):

jedynki = jedynki + "1"

print jedynki # 111 11111

--pix zdefiniował funkcję osobno:

def decimal\_to\_unary(a):

unary = ''

for i in range(a):

unary += '1'

return unary

for i in input:

print(decimal\_to\_unary(i))

---na odwrót, zamień jedynkowy w dziesiętny (czyli policz jedynki):

input= [1111, 1111111]

for i in input:

print len(str(i)) # 4 7

---pix znowu zdefiniował najpiew funkcję:

def unary\_to\_decimal(unary):

return (len(unary))

for i in input:

print(unary\_to\_decimal(str(i)))

------sprawdzanie długości wyrazu i liczby:

wyr = 'xyz'

print len(wyr) # 3

num = 111

print len(str(num)) # 3 # trzeba zamienić int w stringa

print list(str(num)) # [1,1,1] # żeby zamienić w listę, też musi być string

---program, który zamienia cyfry arabskie na rzymskie:

def int\_to\_Roman(num):

val = [1000, 900, 500, 400,100, 90, 50, 40,10, 9, 5, 4,1]

syb = ["M", "CM", "D", "CD","C", "XC", "L", "XL","X", "IX", "V", "IV","I"]

roman\_num = ''

i = 0

while num > 0:

for j in range(num // val[i]):

roman\_num += syb[i]

num -= val[i]

i += 1

return roman\_num

for i in input:

print(int\_to\_Roman(i))

----program zamieniający z systemu arabskiego na cyfry rzymskie:

def int\_to\_Roman(num):

val = [1000, 900, 500, 400,100, 90, 50, 40,10, 9, 5, 4,1]

syb = ["M", "CM", "D", "CD","C", "XC", "L", "XL","X", "IX", "V", "IV","I"]

roman\_num = ' '

i = 0

while num > 0:

for j in range(num // val[i]):

roman\_num += syb[i]

num -= val[i]

i += 1

return roman\_num

for i in input:

print(int\_to\_Roman(i))

----napisz program zamieniający liczby z rzymskiego na arabski:

def toInt(s):

romValues = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}

intValue = 0

for i in range(len(s)):

if i > 0 and romValues[s[i]] > romValues[s[i - 1]]:

intValue += romValues[s[i]] - 2 \* romValues[s[i - 1]]

else:

intValue += romValues[s[i]]

return intValue

np:

MCMLXVI

M

suma = 1000

C>M?

100>1000 nie

suma 1000+100

M>C?

1000>100 tak

1000 - 2\* 100 = 800

suma 1100 + 800 = 1900

L>M?

50>1000 nie

suma +50 =1950

X>L?

10>50 nie

suma+10 = 1960

V>X?

5>10? nie

suma+5=1965

I>V?

1>5? nie

suma+1 = 1966

print toInt("MCMLXVI") # 1996

for i in input:

print(toInt(i))

----zmiana kolejności od tyłu:

val = [1,2,3]

val = val[::-1]

print val[0] # 3

print val # [3, 2, 1]

val = "slowo"

val = val[::-1]

print val # 'owols'

----

number = 90574

print("Binary: " + str(bin(number)))

print("Octal: " + str(oct(number)))

print("Hecadecimal: " + str(hex(number)))

Binary: 0b10110000111001110

Octal: 0260716

Hecadecimal: 0x161ce

---przekonwertuj dziesiętne na szesnastkowe:

410763668

610738024

for i in input:

print hex(i)

0x187bc194

0x24671f68

---przekonwertuj liczby w systemie jedynkowym na dziesiętny, a potem osemkowy:

for liczba in input:

suma = 0

for jedynka in str(liczba): # lub: for jedynka in range (len(str(liczba))):

suma=suma+1

print oct(suma)

(albo, ale wtedy bez for jedynka "in range" bo nie zadziała)

for liczba in input:

suma = 0

for jedynka in str(liczba):

suma=suma+int(jedynka)

print oct(suma)

---------KLASY I OBIEKTY--------------------

class Class:

x = "Test"

object = Class()

print(object.x) # Test

---konstruktor: \_\_init\_\_ zmienna: "self":

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

John = Person("John", 32)

Emily = Person("Emily", 24)

print(John.name)

print(John.age)

print("")

print(Emily.name)

print(Emily.age)

John

32

Emily

24

----dodajemy metodę "introduceYourself". Konstruktor przyjmuje dane (name,age)

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def introduceYourself(self):

print("My name is " + str(self.name) + " and I am " + str(self.age) + " years old.")

John = Person("John", 32)

Emily = Person("Emily", 24)

John.introduceYourself()

Emily.introduceYourself()

My name is John and I am 32 years old.

My name is Emily and I am 24 years old.

---dane są owoc, ilość na magazynie i cena. Napisz program, który obliczy całkowity koszt uzupełnienia braków magazynowych, tak zeby kazdego owocu było po 1000 szt.

Apple 588 3

Pear 664 5

Plum 424 2

Lemon 476 5

Lime 66 3

Grapes 604 3

sumaKosztow = 0

for i in input:

owoc = i.split()[0]

ilosc = int(i.split()[1])

cena = int(i.split()[2])

brakuje = 1000-ilosc

koszt = brakuje\*cena

sumaKosztow +=koszt

print sumaKosztow # 10678

----pix: wersja z utworzeniem klasy:

class Fruit:

def \_\_init\_\_(self, name, amount, price):

self.name = name

self.amount = int(amount)

self.price = int(price)

def cost(self):

return (1000 - self.amount) \* self.price

fruits = []

for i in input:

list = i.split()

fruit = Fruit(list[0], list[1], list[2])

fruits.append(fruit)

sum = 0

for fruit in fruits:

sum += fruit.cost()

print(sum)

-----stwórz klasę Point, która przyjmuje 2 argumenty x i y i metodę distance, któa mierzy odległość między tymi punktami:

input = ["7 5 10 3", "6 5 7 5"]

class Point:

def \_\_init\_\_(self,x,y,x1,y1):

self.x = x

self.y = y

self.x1 = x1

self.y1 = y1

def distance(self): # bez self nie działa!!

odleglosc = ((abs(self.x - self.x1)\*\*2) + (abs(self.y-self.y1)\*\*2))\*\*0.5

return odleglosc # abs w sumie zbędne, bo dajemy do kwadratu..

for i in input:

lista = map(int,i.split())

odl = Point(lista[0],lista[1],lista[2],lista[3]) # odl to nowy obiekt klasy Point

print round(odl.distance(),2)

3.61

1.0

---pix przerzucił drugie x i y (other. x i other.y) do funkcji distance:

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def distance(self, other):

return round(math.sqrt((self.x - other.x) \*\* 2 + (self.y - other.y) \*\* 2), 2)

for i in input:

coordinates = map(int, i.split())

x1, y1, x2, y2 = coordinates[0], coordinates[1], coordinates[2], coordinates[3]

point1 = Point(x1, y1) # pierwszy obiekt klasy Point

point2 = Point(x2, y2) # drugi obiekt klasy Point

print(point1.distance(point2))

----zrobione przez mnie po jakimś czasie:

class Point:

def \_\_init\_\_(self,x,y):

self.x = x

self.y = y

def distance(self,other):

return round(((self.x-other.x)\*\*2+(self.y-other.y)\*\*2)\*\*0.5,2)

for i in input:

lista = map(int,i.split('\t'))

point = Point(lista[0],lista[1])

print point.distance(Vector(lista[2],lista[3])) # gdy zamiast Vector jest Point to też działa

---wypisz imiona tych osób, które zawierają 'a' lub 'A':

input = ['Dagmar', 'Donita', 'Ross', 'Evon']

for i in input:

if ("a" in i) or ('A' in i):

print i # Dagmar Donita

---wersja z class:

class ImieWithA:

def \_\_init\_\_(self,imie):

self.imie = imie

def zawieraA(self):

if ("a" in self.imie) or ('A' in self.imie):

return self.imie

for i in input:

imionko = ImieWithA(i)

if imionko.zawieraA(): # ważne: jeśli bedzie samo print imionko.zawieraA() to będą się wypisywać też None pomiędzy imionami!!!

print i

---zrobione przeze mnie po jakimś czasie:

class Person:

def \_\_init\_\_(self,name):

self.name = name

def imiez\_a(self):

if 'a' in i:

print i

elif 'A' in i: # wyszło tylko dlatego, że jest 'i' w input. Powinno być self.name

print i # i w ogóle tak jak wyżej: if ('a'in i)or('A'in i), jest krótsze i lepsze

def imiez\_a(self):

if 'a' in self.name:

print self.name

elif 'A' in self.name:

print self.name

for i in input:

person = Person(i)

if person.imiez\_a():

person.imiez\_a()

---pix namotał bez sensu:

class Person:

def \_\_init\_\_(self, name):

self.name = name

def \_\_str\_\_(self):

return self.name

people = []

for i in input:

list = i.split("\t") # "\t" to split po tabulatorze, tu jest kompletnie zbędne, patrz niżej

person = Person(list[0]) # wystarczy Person(i)

people.append(person)

(print people >>> nie powstaje lista, tylko coś raczej jak lista potencjałów, niedokończonych obiektów klasy

for p in people:

if "a" in p.name or "A" in p.name:

print(p)

---można to zrobić bez udziwnień (czyli tej listy obiektów):

class Person:

def \_\_init\_\_(self, name):

self.name = name

def \_\_str\_\_(self):

return self.name

for i in input:

person = Person(i)

if "a" in person.name or "A" in person.name:

print person.name

---stwórz klasę Person z polami name i city. Stwórz obiekty tej klasy i wypisz te imiona, które mieszkają w Chicago:

Willis San Jose

Lashawn San Diego

Elvira Chicago

Dona Chicago

class Person:

def \_\_init\_\_(self, name, city):

self.name = name

self.city = city

def chicago(self):

if self.city == "Chicago":

return self.name

for i in input:

list = i.split()

name = list[0]

if len(list) >2:

city = list[1]+" " +list[2]

else:

city = list[1]

obiekt = Person(name,city)

if obiekt.chicago():

print obiekt.chicago() # lub print name

Elvira

Dona

----pix znowu tworzy te listę potencjalnych obiektów:

class Person:

def \_\_init\_\_(self, name, city):

self.name = name

self.city = city

def \_\_str\_\_(self):

return self.name

people = []

for i in input:

list = i.split("\t")

person = Person(list[0], list[1])

people.append(person)

for p in people:

if p.city == "Chicago":

print(p)

---moje zadanko: wyliczyć koszt jednego prania, dane: pojemność butelki, ilosc wlewana do pralki, cena za butelkę:

cennik = ['1 30 71','1 20 93', '2.7 120 19']

for i in cennik:

lista = map(float,i.split()) # w danych są przecinki wiec float a nie int

cena\_litra = lista[2]/lista[0]

print cena\_litra

ilosc\_pran= 1000/lista[1]

print ilosc\_pran # z 1 litra

koszt\_jedn\_prania = cena\_litra/ilosc\_pran

print koszt\_jedn\_prania

71.0 # Amway

33.3333333333 # 33 prania z 1l

2.13 # ponad 2 zł jedno pranie

93.0 # Amway detal

50.0 # 50 prań (po 20 ml)

1.86 # cena 1-ego prania

7.03703703704 # Woolite

8.33333333333 # 8 pań z 1 litra po 120 ml każde

0.844444444444 # 84 grosze 1 pranie

----Python \_\_str\_\_() and \_\_repr\_\_() functions, zwracają stringi, pierwszy lepiej czytelny dla ludzi, z drugiego dane można wykorzystać ponownie przez komp.:

>>> import datetime

>>> now = datetime.datetime.now()

>>> now.\_\_str\_\_()

'2020-12-27 22:28:00.324317'

>>> now.\_\_repr\_\_()

'datetime.datetime(2020, 12, 27, 22, 28, 0, 324317)'

It’s clear from the output that \_\_str\_\_() is more human friendly whereas \_\_repr\_\_() is more information rich and machine friendly and can be used to reconstruct the object. In fact, we can use repr() function with eval() to construct the object.

>>> now1 = eval(repr(now))

>>> now == now1

True

---

class Person:

def \_\_init\_\_(self, person\_name, person\_age):

self.name = person\_name

self.age = person\_age

def \_\_str\_\_(self):

return f'Person name is {self.name} and age is {self.age}'

def \_\_repr\_\_(self):

return f'Person(name={self.name}, age={self.age})'

p = Person('Pankaj', 34)

print(p.\_\_str\_\_())

print(p.\_\_repr\_\_())

Wynik:

Person name is Pankaj and age is 34

Person(name=Pankaj, age=34)

Earlier we mentioned that if we don’t implement \_\_str\_\_ function then the \_\_repr\_\_ function is called. Just comment the \_\_str\_\_ function implementation from the Person class and print(p) will print {name:Pankaj, age:34}

---- dane jest imie, miasto i wiek. Wypisz osoby dorosłe (18+) , które meszkają w miastach jednowyrazowych:

Leanna San Antonio 37

Nakia New York 16

Sheba Albuquerque 53

--moje:

class Person:

def \_\_init\_\_(self, name,age,city):

self.name = name

self.city = city

self.age = int(age)

def adult(self):

if self.age>17 and (" " not in self.city):

return self.name + '\t' + self.city

for i in input:

name = i.split("\t")[0]

age = i.split("\t")[-1]

city = i.split("\t")[1]

person = Person (name, age,city)

if person.adult():

print person.adult() # Sheba Albuquerque

---pix:

class Person:

def \_\_init\_\_(self, name, city, age):

self.name = name

self.city = city

self.age = int(age)

def \_\_str\_\_(self): # metoda, która zwraca string

return self.name + "\t" + self.city

people = []

for i in input:

list = i.split("\t")

person = Person(list[0], list[1], list[2])

people.append(person)

for p in people:

if p.age >= 18 and p.city.count(" ") == 0: # lub ...and (" " not in p.city)

print(p)

---poznajemy metodę klasy "\_\_str\_\_". Służy ona do zwracania stringów. Automatycznie jest ona wywoływana przez funkcje print i str()

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_str\_\_(self):

return "My name is " + str(self.name) + " and I am " + str(self.age) + " years old."

John = Person("John", 32)

Emily = Person("Emily", 24)

print(John) # My name is John and I am 32 years old.

print(Emily) # My name is Emily and I am 24 years old.

EmilyData = str(Emily)

print(EmilyData) # My name is Emily and I am 24 years old.

print Emily.\_\_str\_\_() # My name is Emily and I am 24 years old.

---stworz klasę Point, która przyjmuje dane wartości x i y i wypisz je jako string (x,y):

1 4

3 23

class Point:

def \_\_init\_\_(self,x,y):

self.x = x

self.y = y

def \_\_str\_\_(self):

return "(" + self.x + ", " + self.y +")"

for i in input:

vector = Point(i.split("\t")[0],i.split("\t")[1])

print (vector)

albo zamiast print (vector):

vectorData = str(vector)

print (vectorData) # '(1, 4)' '(3, 23)'

---pix wprowadza funkcję .format():

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_str\_\_(self):

return "({0}, {1})".format(self.x, self.y)

for i in input:

x, y = int(i.split()[0]), int(i.split()[1]) # po co zamieniają na int, żeby zaraz zwracać str

point = Point(x, y)

print(point)

---

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_str\_\_(self):

return "({0.x}; {0.y})".format(self)

def midpoint(self, other):

return Point(float(self.x + other.x) / 2, float(self.y + other.y) / 2)

for i in input:

data = map(int, i.split())

a = Point(data[0], data[1])

b = Point(data[2], data[3])

print(a.midpoint(b)

----

class Count:

def \_\_init\_\_(self,lista):

self.lista = lista

self.suma = 0

self.diff = 2\*lista[0] # dlatego 2\*pierwszy el. listy, bo potem go odejmuje w pętli. Jeśli tu by było sum.diff = 0 to pętla by zaczynała od 0 - list[0] - list[1] -...itd. i list[0] by od razu dostawało minusa. A list[0] musi być pierwszym elementem odejmowania ale jeśli sum.diff = list[0], to wtedy pętla by odejmowała go: list[0] - list[0] - list[1] - ...itd. A tak mamy sum.diff = 2\*list[0] a więc się wszystko pięknie wyzeruje: list[0]+list[0] - list[0] - list[1]-...itd.

def sum(self):

for s in self.lista:

self.suma = self.suma + s

return self.suma

def difference(self):

for d in self.lista:

self.diff = self.diff - d

return self.diff

def \_\_str\_\_(self):

return str(self.sum()) + '\t' + str(self.difference())

for i in input:

list = map(int,i.split('\t'))

obiekt = Count(list)

print (obiekt) # print odpala automatycznie metodę klasy: \_\_str\_\_()

---pix wrzucił też split() do klasy:

class Count:

def \_\_init\_\_(self, numbers):

self.numbers = numbers.split()

def sum(self):

result = 0 # nie ma potrzeby wrzucania tej zmienej do konstruktora \_\_init\_\_ z przydomkiem self.result,bo ona jest tu tylko na chwilę, żeby wyrzucić wynik. Potem wywołujemy tylko funkcję sum() oraz difference()

for i in self.numbers:

result += int(i)

return result

def difference(self):

result = int(self.numbers[0])

for i in range(1, len(self.numbers)):

result -= int(self.numbers[i])

return result

def \_\_str\_\_(self):

return str(self.sum()) + "\t" + str(self.difference())

for i in input:

operation = Count(i)

print(operation)

--- metoda klasy "\_\_int\_\_":

class Person:

def \_\_init\_\_(self, firstName, lastName, age):

self.firstName = firstName

self.lastName = lastName

self.age = age

def \_\_str\_\_(self):

return self.firstName +' ' + self.lastName # jakby tu było self.age to wywali błąd

def \_\_int\_\_(self):

return self.age

John = Person("John", "Brown", 34)

print John # John Brown (jakby nie było def \_\_str\_\_(self) to by wywaliło błąd)

JohnAge = int(John)

print(JohnAge) # 34

print John.\_\_int\_\_() # 34 # przy int\_\_() muszą być nawiasy() a przy \_\_dict\_\_ nie mogą

print John.\_\_dict\_\_ # {'firstName': 'John', 'lastName': 'Brown', 'age': 34}

---stwórz klasę Vector, która obliczy długość

a z podanych liczb(współrzędnych) wg wzoru:

długość wektora = pierwiastek z (x kwadrat + y kwadrat) za pomoca metody klasowej \_\_int\_\_, która ją zaokrągli do liczby całkowitej (trzeba użyć round, bo int tylko utnie po przecinku, np. 1,9 int zmieni w 1 a round w 2).

6 9

6 7

3 8

class Vector:

def \_\_init\_\_(self,x,y):

self.x = int(x)

self.y = int(y)

def \_\_int\_\_(self):

vector = (self.x\*\*2 + self.y\*\*2)\*\*0.5

return int(round(vector))

for i in input:

dlugosc = Vector(i.split()[0],i.split()[1])

print int(dlugosc)

11

9

9

---pix zmienił tylko moje pierwiastkowanie \*\*0.5 na math.sqrt

def \_\_int\_\_(self):

return int(round(math.sqrt(self.x \*\* 2 + self.y \*\* 2)))

---z podanych współrzędnych punktu x i y stwórz klase Point i wylicz odległość od punktu 0,0.

czyli dokładnie to samo co powyzej działa bo wzór to pierwiastek z((x1-x2)do kwadratu + (y1-y2)do kwadratu):

41 38

17 -20

-33 -35

class Point:

def \_\_init\_\_(self,x,y):

self.x = int(x)

self.y = int(y)

def \_\_int\_\_(self):

zero = ((0-self.x)\*\*2+(0-self.y)\*\*2)\*\*0.5

return int(round(zero))

for i in input:

point = Point(i.split()[0],i.split()[1])

print int(point)

56

26

48

----metoda klasy \_\_dict\_\_() zwraca obiekt w postaci słownika, który zawiera nazwy zmiannych obiektu i ich wartości:

class Person:

def \_\_init\_\_(self, firstName, lastName, age):

self.firstName = firstName

self.lastName = lastName

self.age = age

John = Person("John", "Brown", 34)

print(John.\_\_dict\_\_) # {'firstName': 'John', 'lastName': 'Brown', 'age': 34}

---stwórz klasę Person i wypisz za pomocą \_\_dict\_\_ wszystkie dane za pomocą słownika:

Mac Philadelphia 53

Tambra San Diego 64

Hyo New York 73

class Person:

def \_\_init\_\_(self,name,city,age):

self.name = name

self.city = city

self.age = int(age)

for i in input:

person = Person (i.split('\t')[0],i.split('\t')[1],i.split('\t')[2])

print person.\_\_dict\_\_

{'name': 'Mac', 'city': 'Philadelphia', 'age': 53}

{'name': 'Tambra', 'city': 'San Diego', 'age': 64}

{'name': 'Hyo', 'city': 'New York', 'age': 73}

---

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_format\_\_(self, format):

if format == "name":

return self.name

elif format == "age":

return str(self.age)

return "None" # bez tego wywala błąd

John = Person("John", 26)

print("Hi! My name is {0:name} and I'm {0:age} years old.".format(John))

print("Unknown format 'city': {0:city}".format(John)) # nie ma takiego w klasie

Hi! My name is John and I'm 26 years old.

Unknown format 'city': None

---funkcja hasattr(obiekt, atrybut) pozwala na sprawdzenie czy dany obiekt posiada takie pole, jesłi tak to zwraca True jeśli nie to False:

class Animal:

def \_\_init\_\_(self, type):

self.type = type

elephant = Animal("elephant")

print(hasattr(elephant, "type")) #True

print(hasattr(elephant, "age")) #False

jeśli dodam do klasy np. self.age = 3 to to powyżej będze True

---funkcja delattr(obiekt,atrybut) pozwala usunąć pole z klasy.Jesli go nie ma to wywala bład. Należy więc najpierw sprawdzić czy to pole jest używając hasattr():

class Animal:

def \_\_init\_\_(self, type, age):

self.type = type

self.age = age

lion = Animal("lion", 12)

delattr(lion, "age") #delete lion's age attribute

# print(lion.age) error

print(hasattr(lion, "age")) # False

----funkcja del() - usuwanie obiektu:

class Crayon:

def \_\_init\_\_(self, color):

self.color = color

crayon = Crayon("blue") # tworzenie obiektu

print(crayon.color) # blue

del(crayon) # usuwaie obiektu crayon

#print(crayon.color) # error: name 'crayon' is not defined

---

class Crayon:

def \_\_init\_\_(self, color):

self.color = color

def \_\_str\_\_(self):

return "Color: " + str(self.color)

color = ["green", "yellow", "brown", "blue", "red", "black", "white", "pink"]

crayons = []

for i in range(7):

crayon = Crayon(colors[random.randint(0, 7)])

crayons.append(crayon)

for i in crayons:

print(i)

Color: white

Color: black

Color: red

Color: blue

Color: white

Color: brown

Color: red

----------------------------DZIEDZICZENIE KLAS-------------------------------

class Animal:

def \_\_init\_\_(self, name):

self.name = name

class Lion(Animal):

def introduceYourself(self):

print("I'm a lion and my name is " + str(self.name) + ".")

class Elephant(Animal):

def introduceYourself(self):

print("I'm an elephant and my name is " + str(self.name) + ".")

lion = Lion("George")

elephant = Elephant("Sophia")

lion.introduceYourself() # I'm a lion and my name is George.

elephant.introduceYourself() # I'm an elephant and my name is Sophia.

---kiedy konieczne jest zmienienie metody klasy z której dziedziczymy to robimy tzw. przesłanianie metod.

class Lion:

def \_\_init\_\_(self, name):

self.name = name

def introduceYourself(self):

print("I'm a lion and my name is " + str(self.name) + ".")

class Elephant(Lion):

def introduceYourself(self):

print("I'm an elephant and my name is " + str(self.name) + ".")

# (ta sama metoda w dziedziczącej klasie, odziedziczona metoda nie jest używana)

lion = Lion("Lily")

elephant = Elephant("David")

lion.introduceYourself()

elephant.introduceYourself()

---

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Emily = Person("Emily", 27)

print("Name: {0}".format(Emily.name))

print("Age: {0}".format(Emily.age))

---dodane prefixu \_do nazwy zmiennej sprawia że staje się **protected** - **dostępny tylko w klasie i klasach dziedziczących**:

class Person:

def \_\_init\_\_(self, name, age):

self.\_name = name

self.\_age = age

def \_\_str\_\_(self):

return "{0}'s age: {1}.".format(self.\_name, self.\_age)

class Woman(Person):

def \_\_str\_\_(self):

return "Hi! My name is {0} and I'm {1}.".format(self.\_name, self.\_age)

John = Person("John", 36)

print(John) # John's age: 36.

Emily = Woman("Emily", 27)

print(Emily) # Hi! My name is Emily and I'm 27.

------dodane prefixu \_\_do nazwy zmiennej sprawia że staje się **private** - **dostępny tylko wewnątrz klasy**:

class Person:

def \_\_init\_\_(self, name, age):

self.\_\_name = name

self.\_\_age = age

Emily = Person("Emily", 27)

#print("Name: {0}".format(Emily.name)) error

#print("Age: {0}".format(Emily.age)) error

---

class Person:

def \_\_init\_\_(self, name, age):

self.\_\_name = name

self.\_\_age = age

def setAge(self, newAge):

self.\_\_age = newAge

def getAge(self):

return self.\_\_age

Peter = Person("Peter", 36)

print(Peter.getAge()) # 36

Peter.setAge(42) #set Peter's age to 42

print(Peter.getAge()) # 42

---

input = [11,25,23,42]

class Number:

def \_\_init\_\_(self):

self.\_\_value = 0

def addToValue(self,value):

self.\_\_value += value

def getValue(self):

return self.\_\_value

obiekt = Number()

for i in input:

obiekt.addToValue(i)

print obiekt.getValue() # 11 36 59 101

---

class Person:

def \_\_init\_\_(self, name, age):

self.\_\_name = name

self.\_\_age = age

def getValue(self, valueName):

if valueName == "name":

return self.\_\_name

elif valueName == "age":

return self.\_\_age

def setValue(self, valueName, value):

if valueName == "name":

self.\_\_name = value

elif valueName == "age":

self.\_\_age = value

Lily = Person("Lily", 35)

print("Name: {0}, age: {1}.".format(Lily.getValue("name"), Lily.getValue("age")))

# Name: Lily, age: 35.

Lily.setValue("age", 22)

print("Name: {0}, age: {1}.".format(Lily.getValue("name"), Lily.getValue("age")))

# Name: Lily, age: 22.

Lily.setValue('name',"Aga")

print ('New name: {0}, age: {1}.'.format(Lily.getValue('name'),Lily.getValue('age')))

# New name: Aga, age: 22.

---

class Person:

def \_\_init\_\_(self, name, age):

self.\_\_name = name

self.\_\_age = age

def getAge(self):

return self.\_\_age

def setAge(self, value):

self.\_\_age = value

age = property(getAge, setAge) # należy do klasy

Lily = Person("Lily", 35)

print(Lily.age) # tu musi być age czy inna nazwa jaka jest w klasie

Lily.age1 = 22 # może być age, aage1 czy xyz, działa... byle w print była taka sama nazwa

print(Lily.age1)

Lily.xyz = '42a'

print (Lily.xyz) # 42a nie wiem o co chodzi, bo tekst też wchodzi

---przecieżanie operatorów, czyli stworzenie metod, które np. umożliwią dodawanie albo odejmowanie dwóch obiektów tej samej klasy. Tutaj metodami \_\_add\_\_ dodawanie i \_\_sub\_\_ odejmowanie:

class Number:

def \_\_init\_\_(self, value):

self.value = value

# method to handle addition operator

def \_\_add\_\_(self, other):

return self.value + other.value

# method to handle substraction operator

def \_\_sub\_\_(self, other):

return self.value - other.value

six = Number(6)

four = Number(4)

print(six + four) # 10

print(six - four) # 2

ten = six + four #it creates a variable of type int, no Number object

print(ten) # 10

two = six - four #it creates a variable of type int, no Number object

print(two) # 2

---stwórz klase Point, która przyjmie wartosci x i y a następnie wypisz dla podanych par punktów watości ich dodawania i odejmowania w postaci (x, y) oddzielone tabulatorem:

x y x1 y1

13 12 19 14

12 17 19 16

19 11 14 11

class Point:

def \_\_init\_\_(self, x,y):

self.x = x

self.y = y

def \_\_add\_\_(self,other):

addx = self.x + other.x

addy = self.y + other.y

return ("({0}, {1})".format(addx,addy))

def \_\_sub\_\_(self,other):

subx = self.x - other.x

suby = self.y - other.y

return ("({0}, {1})".format(subx,suby))

for i in input:

points = map(int,i.split())

punkt1 = Point(points[0],points[1])

punkt2 = Point(points[2],points[3])

print ("{0} {1}".format(punkt1+punkt2,punkt1-punkt2))

(x+x1, y+y1) (x-x1, y-y1)

(32, 26) (-6, -2)

(31, 33) (-7, 1)

(33, 22) (5, 0)

---kolejne metody służące przeciążania operatoró to **\_\_mul\_\_** mnożenie i operator \*, oraz **\_\_div\_\_** dzielenie i operator /.

class Number:

def \_\_init\_\_(self, value):

self.value = float(value)

# method to handle multiplication operator

def \_\_mul\_\_(self, other):

return self.value \* other.value

# method to handle division operator

def \_\_div\_\_(self, other):

return self.value / other.value

six = Number(6)

four = Number(4)

print(six \* four) # 24.0

print(six / four) # 1.5

---stwórz klasę Vector, któa przyjmie wartości x i y a następnie wypisz je przemnozone przez trzecia podaną liczbę w postaci wektora [x, y]

27 20 7

23 22 7

29 21 4

class Vector:

def \_\_init\_\_(self,x,y):

self.x = x

self.y = y

def \_\_mul\_\_(self,multi):

return "[{0}, {1}]".format(self.x\*multi,self.y\*multi)

for i in input:

points = map(int,i.split())

vector = Vector(points[0],points[1])

print vector\*points[2]

[189, 140]

[161, 154]

[116, 84]

---pix wrzucił forma to do osobnej metody klasy: \_\_str\_\_:

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_str\_\_(self):

return "[{0}, {1}]".format(self.x, self.y)

def \_\_mul\_\_(self, value):

self.x = self.x \* value

self.y = self.y \* value

for i in input:

x, y = int(i.split()[0]), int(i.split()[1])

vector = Vector(x, y)

vector \* int(i.split()[2])

print(vector) # po przemnożeniu vector zmiania wartość

---

Pear 1.0 17

Lemon 1.5 20

Plum 1.5 20

class Product:

def \_\_init\_\_(self,name,price):

self.name = name

self.price = price

def \_\_mul\_\_(self,amount):

self.price = self.price \* amount

def \_\_str\_\_ (self):

return "{0} {1}".format(self.name,float(self.price))

for i in input:

product = Product(i.split()[0],float(i.split()[1]))

product \* float(i.split()[2]) # tu może być też int, można int z float pomnożyć

print (product)

Pear 17.0

Lemon 30.0

Plum 30.0

---pix właściwie zrobił tak samo. Detale zaznaczone:

class Product:

def \_\_init\_\_(self, name, price):

self.name = name

self.price = float(price)

def \_\_mul\_\_(self, quantity):

self.price = self.price \* quantity

def \_\_str\_\_(self):

return "{0}**\t**{1}".format(self.name,self.price) tu już nie trzeba float bo jest wcześniej

for i in input:

data = i.split()

product = Product(data[0], data[1])

product \* int(data[2])

print(product)

---przeciążanie kolejnych operatorów, aby móc porównywać dwa obekty: < - **\_\_lt\_\_** (less than) i

> - **\_\_gt\_\_** (greater than):

class Number:

def \_\_init\_\_(self, value):

self.value = value

#method to handle < operator

def \_\_lt\_\_(self, other):

return self.value < other.value

#method to handle > operator

def \_\_gt\_\_(self, other):

return self.value > other.value

six = Number(6)

four = Number(4)

print(six > four) # True

print(four > six) # False

---stwórz klasę Vector z dwoma polami x i y, w której będzie metoda sortująca a nastepnie utwórz obiekt i wypisz współrzedne kolejnych wektorów po kolei według ich długości rosnąco:

34 36

-40 -31

27 22

class Vector:

def \_\_init\_\_(self, x, y):

self.x = int(x)

self.y = int(y)

def distance(self):

leng = (self.x\*\*2 +self.y\*\*2)\*\*0.5

return leng

def \_\_gt\_\_(self,other):

return self.distance() > other.distance()

def \_\_str\_\_(self):

return "[{0}, {1}]".format(self.x,self.y)

wektory = []

for i in input:

wektor = Vector(int(i.split()[0]),int(i.split()[1]))

wektory.append(wektor)

wektory.sort() # **sort()** działa tylko jeśli w klasie istnieje metoda \_\_lt\_\_ albo \_\_gt\_\_

for i in wektory:

print i

[27, 22]

[34, 36]

[-40, -31]

---pix wrzucił distance do \_\_init\_\_ i użył \_\_lt\_\_ zamiast \_\_gt\_\_ (ale zadziałało tak samo):

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

self.distance = math.sqrt(self.x \*\* 2 + self.y \*\* 2)

def \_\_str\_\_(self):

return "[{0}, {1}]".format(self.x, self.y)

def \_\_lt\_\_(self, other):

return self.distance < other.distance

vectors = []

for i in input:

x, y = int(i.split()[0]), int(i.split()[1])

vector = Vector(x, y)

vectors.append(vector)

vectors.sort()

for i in vectors:

print(i)

---stwórz klasę People z polami name i age, posortuj wg wieku malejąco i wypisz oddzielone tabulatorem imie i wiek:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_lt\_\_(self,other):

return self.age < other.age

def \_\_str\_\_(self):

return "{0}\t{1}".format(self.name,self.age)

people=[]

for i in input:

person = Person(i.split("\t")[0],int(i.split('\t')[1]))

people.append(person)

people.sort()

people = people[::-1]

for i in people:

print i

PS. Odwrócenie w metodzie klasy znaku > też działa (choć może nie jest poprawne?) ale jeśli dwie osoby maja wiek taki sam, to inaczej wypisuje się kolejność niż kiedy użyję people = people[::-1].

np Joanna 42 Zosia 42, zamiast Zosia 42 Joanna 42

def \_\_lt\_\_(self,other):

return self.age **>** other.age

---pix zamiast people = people[::-1]wstawili people.sort(reverse = True)

---kolejne operatory, które można przeciążyć to **\_\_eq\_\_** czyli **==** oraz **\_\_ng\_\_** czyli **!=** :

class Number:

def \_\_init\_\_(self, value):

self.value = value

#method to handle == operator:

def \_\_eq\_\_(self, other):

return self.value == other.value

#method to handle != operator:

def \_\_ne\_\_(self, other):

return self.value != other.value

six = Number(6)

four = Number(4)

print(six == four) # False

print(four != six) # True

---stworz klasę Vector która wylicza długość wektora na podstawie podanych współrzędnych (x i y) a potem porównaj czy podane pary współrzędnych są równej długości (x,y)(x1,y1):

8 5 7 7

7 8 7 8

6 8 8 6

class Vector:

def \_\_init\_\_(self,x,y):

self.x = x

self.y = y

self.leng = math.sqrt((self.x)\*\*2+(self.y)\*\*2)

def \_\_eq\_\_ (self,other):

return self.leng == other.leng

for i in input:

list = map(int,i.split())

vec1 = Vector(list[0], list[1])

vec2 = Vector(list[2],list[3])

print (vec1 == vec2)

False

True

True

-----------------SORTOWANIE, ALGORYTMY SORTUJĄCE------------

---sortowanie bąbelkowe:

list = [3, 1, 2, 4, 0]

print('List before bubble sorting:')

print(str(list))

def bubble\_sort(list):

for i in range(len(list) - 1, 0, -1):

for j in range(i):

if(list[j] > list[j + 1]):

list[j], list[j + 1] = list[j + 1], list[j]

print (list)

bubble\_sort(list)

print('List after bubble sorting:')

print(str(list))

List before bubble sorting:

[3, 1, 2, 4, 0]

[1, 3, 2, 4, 0]

[1, 2, 3, 4, 0]

[1, 2, 3, 0, 4]

[1, 2, 0, 3, 4]

[1, 0, 2, 3, 4]

[0, 1, 2, 3, 4]

List after bubble sorting:

[0, 1, 2, 3, 4]

----sortowanie przez wstawianie:

list = [10, 3, 7, 8, 1, 6, 2, 9, 4, 5, 0]

print('List before insertion sort:')

print(str(list))

def insert\_sorting(list):

for i in range(1, len(list)):

key = list[i]

j = i - 1

while(j >= 0 and list[j] > key):

list[j + 1] = list[j]

j -= 1

list[j + 1] = key

print(list)

insert\_sorting(list)

print('List after insertion sort:')

print(list)

---sortowanie przez scalanie:

list = [10, 3, 7, 8, 1, 6, 2, 9, 4, 5, 0]

print('List before merge sort:')

print(str(list))

def merge\_sort(list):

if(len(list) > 1):

mid = len(list) // 2

left\_half = list[:mid]

right\_half = list[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

while(i < len(left\_half) and j < len(right\_half)):

if(left\_half[i] < right\_half[j]):

list[k] = left\_half[i]

i+=1

else:

list[k] = right\_half[j]

j+=1

k+=1

while(i < len(left\_half)):

list[k] = left\_half[i]

i+=1

k+=1

while(j < len(right\_half)):

list[k] = right\_half[j]

j += 1

k += 1

merge\_sort(list)

print('List after merge sort:')

print(str(list))

---moje sortowanie - za pomocą klasy:

list = [10, 3, 7, 8, 1, 6, 2, 9, 4, 5, 0]

class Sort:

def \_\_init\_\_(self,x):

self.x = int(x)

def \_\_gt\_\_(self,other):

return self.x > other.x

def \_\_str\_\_(self):

return str(self.x)

obiekty=[]

for i in list:

obiekt = Sort(i)

obiekty.append(obiekt)

obiekty.sort()

lista = []

for i in obiekty:

lista.append(**str**(i)) # bez **str** orzymujemy listę obiektów(potencjałów) czyli bełkot

print **map(int**,lista) # powyzej int nie zadziała, więc trzeba zmapować int na listę, bo inaczej otrzymamy listę stringów ['0','1','2',...]

**---wersja lepsza i bez dodatkowego mieszania i tworzenia listy:**

list = [10, 3, 7, 8, 1, 6, 2, 9, 4, 5, 0]

class Sort:

def \_\_init\_\_(self,x):

self.x = int(x)

def \_\_gt\_\_(self,other):

return self.x > other.x

def **\_\_str\_\_(**self):

return str(self.x)

obiekty=[]

for i in list:

obiekt = Sort(i)

obiekty.append(**str**(obiekt)) # str wywołą metodę \_\_str\_\_ która zwróci nam stringi

obiekty.sort()

print **map(int**,obiekty)

**---a tu wersja jeszcze lepsza - do klasy zamiast metody \_\_str\_\_ wstawiamy metodę \_\_int\_\_:**

list = [10, 3, 7, 8, 1, 6, 2, 9, 4, 5, 0]

class Sort:

def \_\_init\_\_(self,x):

self.x = int(x)

def \_\_gt\_\_(self,other):

return self.x > other.x

def \_\_int\_\_(self):

return int(self.x)

obiekty=[]

for i in list:

obiekt = Sort(i)

obiekty.append(**int**(obiekt)) # tu wystarczy wstawić int żeby otrzymać postać int zamiast bełkotu

obiekty.sort()

print (obiekty) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

zamiana zdania na listę:

sentence = "The quick brown fox".upper()

sentence = list(sentence)

print sentence

['T', 'H', 'E', ' ', 'Q', 'U', 'I', 'C', 'K', ' ', 'B', 'R', 'O', 'W', 'N', ' ', 'F', 'O', 'X']

----print list(sentence) też by wypisało jw. ale potem print sentence by wypisało:

print list(sentence) # ['T', 'H', 'E', ' ', 'Q', 'U', 'I', 'C', 'K', ' ', 'B', 'R', 'O', 'W', 'N', ' ', 'F', 'O', 'X']

print sentence # THE QUICK BROWN FOX

--bo nie zmianiamy samego sentence tylko każemy mu sę w taki sposów wydrukować

----szyfrowanie - zamieniamy miejscami każde dwie litery:

sentence = "The quick brown fox jumps over a lazy dog".upper()

for i in range(0, len(sentence) - 1, 2):

sentence = list(sentence)

sentence[i], sentence[i + 1] = sentence[i + 1], sentence[i]

sentence = "".join(sentence)

print(sentence) # HT EUQCI KRBWO NOF XUJPM SVOREA L ZA YODG

---zamiana miejscami co trzecią literę:

sentence = "The quick brown fox".upper()

for i in range(0, len(sentence) - 1,3):

sentence = list(sentence)

sentence[i], sentence[i + 1],sentence[i+2] = sentence[i+2],sentence[i + 1], sentence[i]

sentence = "".join(sentence)

print(sentence)

EHT QUICK BROWN FOX

EHTUQ ICK BROWN FOX

EHTUQ KCI BROWN FOX

EHTUQ KCIRB OWN FOX

EHTUQ KCIRB NWO FOX

EHTUQ KCIRB NWOOF X

--w podanych wyrazach zamień miejscami co dwie litery, a potem wypisz je duzymi literami:

fig

blueberry

watermelon

for wyraz in input:

wyraz = wyraz.upper()

for j in range(0,len(wyraz)-1, 2):

wyraz = list(wyraz)

wyraz[j],wyraz[j+1]=wyraz[j+1], wyraz[j]

print ''.join(wyraz) # bez ''.join byłaby lista ['A', 'W', 'E', 'T', 'M', 'R', 'L', 'E', 'N', 'O']

IFG

LBEUEBRRY

AWETMRLENO

---odszyfrowanie to dokładnie ten sam kod:

input = ["HT EUQCI KRBWO NOF XUJPM SVORET EHL ZA YODG"]

zdanie = input[0]

for i in range(0,len(zdanie)-1,2):

zdanie = list(zdanie)

zdanie[i],zdanie[i+1]=zdanie[i+1],zdanie[i]

zdanie = ''.join(zdanie)

print zdanie # THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

print zdanie.lower() # the quick brown fox jumps over the lazy dog

----zapisz podane elementy do listy, a następnie usuń z niej duplikaty i ją wypisz:

12

5

14

12

14

list = input

for i in range(len(list)):

for j in range(i+1,len(list)-2):

if list[i] == list[j]:

del list[j]

print list # [12, 5, 14]

---pix zamiast kasować z istniejącej listy to stworzył i nadpisywał nową. Zamiast sprawdzać czy elementy są takie same to sprawdzał czy są już na nowej liście (if not i in newList):

list = input

newList = []

for i in list:

if not i in newList: # może też być: if i not in newList

newList.append(i)

print(newList)

---niestety rozwiązanie pix jest znacznie lepsze, moje skraca listę i przez to jej długość:

list = [77, 33, 11, 22, 88, 99, 99, 22, 33, 55]

list.sort() # okazuje się że jest sposob na sortowanie listy bez skomplikowanych działań i że nie trzeba rozpisywać klasy, żeby użyć funkcji **.sort()**

print list # [11, 22, 22, 33, 33, 55, 77, 88, 99, 99]

new = []

for i in list:

if i not in new:

new.append(i)

print new # [11, 22, 33, 55, 77, 88, 99]

for i in range (len(list)):

for j in range(i+1,len(list)-1): # źle, bo długość listy się zmienia, bo usuwamy jej elementy

if list[i]==list[j]:

del list[j]

print list # [11, 22, 33, 55, 77, 88, 99, 99]

----wyszukiwanie binarne:

----wypisz wszystkie liczby z danego zakresu, podzielne przez podane liczby

144 177

2

5

10

input = ['144 177','',2,5,10]

zakres\_min = int(input[0].split()[0])

zakres\_max = int(input[0].split()[1])

podzielne = []

for i in range(zakres\_min,zakres\_max+1):

for j in range(2,len(input)):

if i%input[j] ==0:

podzielne.append(i)

podzielne\_short=[]

for liczba in podzielne:

if liczba not in podzielne\_short:

podzielne\_short.append(liczba)

print podzielne\_short # [144, 145, 146, 148, 150, 152, 154, 155, 156, 158, 160, 162, 164, 165, 166, 168, 170, 172, 174, 175, 176]

---wersja druga:

wypisz liczby z danego zakresu, podzielne przez każdą z podanych liczb:

zakres\_min = int(input[0].split()[0])

zakres\_max = int(input[0].split()[1])

podzielne = []

for i in range(zakres\_min,zakres\_max+1):

if i%input[2] ==0 and i%input[3]==0 and i % input[4]==0:

podzielne.append(i)

podzielne\_bez\_dubli=[]

for liczba in podzielne:

if liczba not in podzielne\_bez\_dubli:

podzielne\_bez\_dubli.append(liczba)

print (podzielne\_bez\_dubli) # [150, 160, 170]

--wypisane jedno pod drugim:

for i in podzielne\_bez\_dubli:

print i

---pix:

dividers = [input[i] for i in range(2, len(input))]

for i in range(int(input[0].split()[0]), int(input[0].split()[1]) + 1):

for j in dividers:

if i % j != 0:

break

else:

print(i)

---trochę powtorki: to robi to samo...:

input=[0,11,22,33,44,55]

dividers = [input[i] for i in range(2, len(input))]

print dividers # [22,33,44,55]

--...robi to samo co to:

div = []

for i in range(2, len(input)):

div.append(input[i])

print div # [22,33,44,55]

----napisz funkcję rozkładającą liczbę na czynnki pierwsze:

def rozkl\_na\_pierwsze(x):

pierwsza = 2

while pierwsza <=x:

if x % pierwsza == 0:

x = x/pierwsza

print pierwsza

else:

pierwsza+=1

rozkl\_na\_pierwsze(44) # 2 2 11

---z podanych liczb w input wylicz czynniki pierwsze, wypisując je jaki listę:

input = [44,33]

def rozkl\_na\_pierwsze(x):

pierwsza = 2

lista=[]

while pierwsza <=x:

if x % pierwsza == 0:

x = x/pierwsza

lista.append(pierwsza)

else:

pierwsza+=1

return lista

for i in input:

print rozkl\_na\_pierwsze(i)

[2, 2, 11]

[3, 11]

----pix

for i in input:

n = int(i)

primeFactors = []

for j in range(2, n + 1):

while not n % j:

primeFactors.append(j)

n /= j

if n == 1:

break

print(primeFactors)

input = [77,79,70]

def dzielniki(x):

dzielnik=2

dzielniki=[]

while dzielnik<=x:

if x % dzielnik == 0:

dzielniki.append(dzielnik)

x=x/dzielnik

else:

dzielnik+=1

return dzielniki

start=input[0]

end=input[1]

num=input[2]

print 'dzielniki 70 ',dzielniki(num)

for i in range(start,end+1):

print "to jest i ", i

print "dzielniki i " , dzielniki(i)

for j in dzielniki(num):

temp=''

if j in dzielniki(i):

break

else:

temp= True

if temp==True:

print i

dzielniki 70 [2, 5, 7]

to jest i 77

dzielniki i [7, 11]

to jest i 78

dzielniki i [2, 3, 13]

to jest i 79

dzielniki i [79]

79 #(Alleluja, tego szukałam)

----input = [87,101,90]

def dzielniki(x):

dzielnik=2

dzielniki=[]

while dzielnik<=x:

if x % dzielnik == 0:

dzielniki.append(dzielnik)

x=x/dzielnik

else:

dzielnik+=1

return dzielniki

start=input[0]

end=input[1]

num=input[2]

#print 'dzielniki 70 ',dzielniki(num)

for i in range(start,end+1):

#print "to jest i ", i

#print "dzielniki i " , dzielniki(i)

for j in dzielniki(num):

temp=''

if j in dzielniki(i):

break

else:

temp= True # to jest po to, żeby nie dawać tu print i, bo wydrukuje tyle razy ile mamy j

if temp==True: # a tak to tylko raz druknie i

print i # 89 91 97 101

---pix (nic z tego nie kapuję). Wzór jest że a i b są względnie pierwsze, jeśli dla a i b istnieje x i y, że a\*x+b\*y = 1

a = input[0]

b = input[1]

p = input[2]

for i in range(a, b + 1):

ax = i

bx = p

while bx:

t = bx

bx = ax % bx

ax = t

if ax == 1:

print(i)

---odszyfruj podane wyrazy zaszyfrowane szyfrem Cezara, ilość przesunięcia jest podana jako pierwsza:

input = [3,'ohprq','eoxhehuub','udvsehuub','eodfnehuub','judsh,'ilj']

move = input[0]

texts = input[1::]

for i in texts:

word = ""

for j in i:

if (ord(j) >= 97 + move):

word += chr(ord(j) - move)

else:

word += chr(ord(j) + 26 - move)

print(word)

---moje na logikę - niezbyt wychodzi, pozostają jakieś dodatkowe literki na końcu wyrazu, nie wiem skąd...

alfabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']

przesuniecie = int(input[0])

for i in range(1,len(input)):

wyraz = list(input[i])

odszyfrowany=[]

for literka in wyraz:

for i in range(len(alfabet)):

if literka==alfabet[i]:

if i>=przesuniecie:

literka=alfabet[i-przesuniecie]

else:

literka=alfabet[(len(alfabet)-1)-(przesuniecie-i)]

if len(odszyfrowany) < len(wyraz):

odszyfrowany.append(literka)

print "".join(odszyfrowany)

lemon

blueberry

raspberry

blackberry

grape

fig

---program ktory wyliczy ilosc wszystkich mozliwych wariacji podanych wyrazów(ciagow znaków) bez powtórzeń(permutacji):

vHSrIf

kdwXR

hWXcTSAO

def factorial(n):

if n == 1:

return 1

else:

return n \* factorial(n - 1)

for i in input:

print(factorial(len(i)))

720

120

40320

---znajdowanie indeksu:

input=[0,11,22,33,44,55]

print input.index(22) # 2

---szyfr podstawieniowy polega na tym, ze każda litera jest zastąpiona inną wg. klucza, np. A zamieniamy na C a B na A:

string = "ABB BBC CCB ABC"

base = "ABC"

key = "CAB"

crypted = ""

for i in string:

if i not in base:

crypted += i # jeśli literki nie ma w bazie to nie szyfrujemy tylko przepisujemy

else:

crypted += key[base.index(i)] # a jeśli jest to podmieniamy literkę na z "key"

print("Before: %s" % string) # Before: ABB BBC CCB ABC

print("After: %s" % crypted) # After: CAA AAB BBA CAB

----zaszyfruj podane ciągi (stringi) metodą podstawieniową. W pierwszym elemencie jest alfabet a w drugim klucz:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

KAPIZBTEUOYVLGDHQSRCJWMNXF

XLC XMLRW TPINDRIOXIUUXBKQRQ DASLMNULBFILDIBFLUICJKOQ BETF DPEYLYGVANVIL AS

PRI UTBDUITSUKJYBTHWNOAPDGRZXX TQVI AKJZHWFYGSL VTX

base = input[0]

key = input[1]

stringi = [input[i] for i in range(2,len(input))]

for sentence in stringi:

crypted = ""

for i in sentence:

if i not in base:

crypted += i

else:

crypted += key[base.index(i)]

print crypted

OXY OWXLK JQANELAZOAPPODBCLC EGHXWNPXDSAXEADSXPAYFBZC DRJS EQRUXUMIGNIAX GH

QLA PJDEPAJHPBFUDJVKNZGQEMLTOO JCIA GBFTVKSUMHX IJO

----------------------------SŁOWNIKI-----------------------------------------

słownik zamiast indeksu ma **klucz**(hasło), którym może być napis, liczba, lista, np. **Name, Version, Year** a **obiekty** tego **słownika** to **PixBlox, Desktop, 2019.**

dictionary = { 'Name' : 'PixBlocks', 'Version' : 'Desktop', 'Year' : 2019 }

print(dictionary) # {'Version': 'Desktop', 'Name': 'PixBlocks', 'Year': 2019}

-----------

input = ['Fruit','Pear','Vegetable','Lettuce']

dict={}

dict[input[0]] = input[1]

dict[input[2]] = input[3]

print dict # {'Vegetable': 'Lettuce', 'Fruit': 'Pear'}

# pix piszczy, że ma być odwrotnie { 'Fruit': 'Pear', 'Vegetable': 'Lettuce'} jedyne słuszne rozwiązanie:

dictionary = { input[0] : input[1], input[2] : input[3] }

print(dictionary)

----

dictionary = { 'Name' : 'PixBlocks', 'Version' : 'Desktop', 'Year' : 2019 }

x = dictionary['Version']

print(x) # Desktop wypisanie obiektu slownika szukając po kluczu

dictionary['Year'] = 2010 # zmiana wartości

print(dictionary) # {'Version': 'Desktop', 'Name': 'PixBlocks', 'Year': 2010}

dictionary['User'] = 'Student' # wprowadzenie nowego klucza i obiektu

print(dictionary) # {'User': 'Student', 'Version': 'Desktop', 'Name': 'PixBlocks', 'Year': 2010}

----instrukcja warunkowa if szuka klucza w słowniku, po obiekcie nie znajdzie:

dictionary = { 'Name' : 'PixBlocks', 'Version' : 'Desktop', 'Year' : 2019 }

if('Year' in dictionary):

print('Yes, one of the keys is named \'Year\'.') # Yes, one of the keys is named 'Year'.

if('Desktop' in dictionary):

print('Yes, one of the keys is named \'Year\'.') # nie znajdzie i nie wypisze

print('Length of our dictionary is: ' + str(len(dictionary))) # Length of our dictionary is: 3

---długość słownika to ilość kluczy (klucz i jego wartość są traktowane jak jedno)

dictionary = { 'Name' : 'PixBlocks', 'Version' : 'Desktop', 'Year' : 2019 }

dictionary.pop('Name')

print(dictionary) # {'Version': 'Desktop', 'Year': 2019}

del dictionary['Year']

print(dictionary) # {'Version': 'Desktop'}

dictionary.clear()

print(dictionary) # {}

# del dictionary

# print(dictionary) # Name 'dictionary' is not defined. (Because we deleted it.)

----**metoda słownik.copy()** oraz funkcja **dict(słownik)** kopiują słownik - ten sam rezultat:

dictionary = { 'Name' : 'PixBlocks', 'Version' : 'Desktop', 'Year' : 2019 }

first\_copy = dictionary.copy()

print(first\_copy) # {'Version': 'Desktop', 'Name': 'PixBlocks', 'Year': 2019}

second\_copy = dict(dictionary)

print(second\_copy) # {'Version': 'Desktop', 'Name': 'PixBlocks', 'Year': 2019}

---sortowanie słownika - wg klucza i wg. wartości:

prices = {"Apple": 2.00, "Banana": 1.00, "Orange": 1.50, "Cantaloupe": 4.00, "Grapes": 0.50}

print("Sorting by Keys:") # Sorting by Keys:

print(sorted(prices.iterkeys())) # ['Apple', 'Banana', 'Cantaloupe', 'Grapes', 'Orange']

print(sorted(prices)) # samo sorted(prices) robi to samo co to powyżej sorted(prices.iterkeys())

print("\nSorting by Values:") # Sorting by Values:

print(sorted(prices.iteritems(), key = lambda x : x[1]))

# [('Grapes', 0.5), ('Banana', 1.0), ('Orange', 1.5), ('Apple', 2.0), ('Cantaloupe', 4.0)]

bez key=lambda x : x[1] czyli: print(sorted(prices.iteritems())) sortuje klucze i wypisuje z wartościami

# [('Apple', 2.0), ('Banana', 1.0), ('Cantaloupe', 4.0), ('Grapes', 0.5), ('Orange', 1.5)]

----stwórz słownik i wypisz osoby mieszkające w New York City posortowane alfabetycznie:

input = ['Lashawn Philadelphia', 'Willis New York City', 'Leanna Phoenix', 'Aurelia New York City']

dict = {}

nyc = {}

for i in input:

dict[i.split('\t')[0]] = i.split('\t')[1]

if i.split('\t')[1]=='New York City':

nyc[i.split('\t')[0]] = i.split('\t')[1]

print sorted(nyc.iterkeys()) # ['Aurelia', 'Willis']

---wersja z wypisywaniem imion jedno po drugim:

input = ['Lashawn Philadelphia','Willis New York City','Leanna Phoenix','Aurelia New York City']

dict = {}

nyc = []

for i in input:

dict[i.split('\t')[0]] = i.split('\t')[1]

if i.split('\t')[1]=='New York City':

nyc.append(i.split('\t')[0])

for i in sorted(nyc):

print i

albo:

nyc.sort()

for i in nyc:

print i

Aurelia

Willis

---pix zrobił to prościej:

dictionary = {}

for i in input:

list = i.split('\t')

dictionary[list[0]] = list[1]

for i in sorted(dictionary):

if dictionary[i] == "New York City":

print(i)

---wypisz średnią wieku osób mieszkających w Phoenix:

Tambra Dallas 7

Sharri Phoenix 94

Agnes Los Angeles 25

Lekisha New York City 36

Anibal Phoenix 51

dict={}

sum = []

for string in input:

list = string.split('\t')

if list[1] == 'Phoenix':

sum.append(float(list[2]))

wynik = 0

for i in sum:

wynik +=i

print wynik/len(sum) # 72.5

---pix pojechał po bandzie - myślałam, że coś ze słownikami zaczarują a oni klasę wypakowali, nie wiem po co tyle kodu, ale ok, powtórka:

class Person:

def \_\_init\_\_(self, name, city, age):

self.name = name

self.city = city

self.age = int(age)

def \_\_str\_\_(self):

return self.name

people = []

for i in input:

list = i.split('\t')

person = Person(list[0],list[1],list[2])

people.append(person)

---(jeśli w tym momencie wypiszę "print people" to wywali jakieś krzaczki, bo str z klasy dopiero działa jeśli bym zrobiła print na każdym obiekcie z listy(poniżej), albo tak:

people.append(str(person))

print people # ['Tambra', 'Sharri', 'Agnes', 'Lekisha', 'Anibal']

albo:

for p in people:

print p # wypisze jedno pod drugim: 'Tambra' 'Sharri' 'Agnes' 'Lekisha' 'Anibal'

----

sum = 0.0

n = 0

for p in people:

if(p.city == 'Phoenix'):

sum += p.age

n += 1

print(sum/n)

-----podane przedmioty pierwszy to pojemnik a reszta to przedmioty w pojemniku, stwórz słownik zawierający pojemnik jako key a listę przedmiotów jako obiekt pod postacią listy i wypisz ten słownik:

pouch flint twine gemstone

backpack hatchet dagger bedroll

pocket seashell strange berry biscuit

dict={}

for i in input:

list=i.split('\t')

key = list[0]

items = []

for j in range(1,len(list)):

items.append(list[j])

dict[key] = items

print dict # {'backpack': ['hatchet', 'dagger', 'bedroll'], 'pouch': ['flint', 'twine', 'gemstone'], 'pocket': ['seashell', 'strange berry', 'biscuit']}

----pix zrobił prościej bo zna magiczną sztuczkę **[1:]**:

inventory = {}

for i in input:

list = i.split('\t')

inventory[list[0]] = list**[1:] # to robi listę elementów listy od 1 do końca!!!**

inventory[list[0]] = [list[i] for i in range(1,len(list))] # może być też tak zapisane

print inventory

---ćwiczonka:

lista = ['pyk','la','lal','lala']

print lista[1:] ['la', 'lal', 'lala']

print lista[-1:] ['lala']

print lista[::-1] ['lala', 'lal', 'la', 'pyk']

print lista [:1] ['pyk']

print lista [:-1] ['pyk', 'la', 'lal']

lista = ['pyk','la','lal','lala','lalali']

print lista[2:] ['lal', 'lala', 'lalali'] od indexu 2 włacznie do końca

print lista[-2:] ['lala', 'lalali'] od indexu -2 włacznie (2 od końca) do końca

print lista[::-2] ['lalali', 'lal', 'pyk'] od od końca do początku co drugi

print lista[::2] ['pyk', 'lal', 'lalali'] od początku do końca co drugi

print lista [:2] ['pyk', 'la'] od początku do indexu 2 (bez indexu 2)

print lista [:-2] ['pyk', 'la', 'lal'] od początku do indexu -2 (2 od końca, czyli 3) bez -2

lista = ['0','1','2','3','4','5']

print lista[2:] ['2', '3', '4', '5']

print lista[-2:] ['4', '5']

print lista[::-2] ['5', '3', '1']

print lista[::2] ['0', '2', '4']

print lista [:2] ['0', '1']

print lista [:-2] ['0', '1', '2', '3']

lista = ['0','1','2','3','4','5', '6']

print lista[3:] ['3', '4', '5', '6']

print lista[-3:] [ '4', '5', '6']

print lista[::-3] [ '6', '3', '0']

print lista[::3] ['0', '3', '6']

print lista [:3] ['0', '1', '2']

print lista [:-3] ['0', '1', '2', '3']

---iteracja po słowniku, tyle, że słownik nie zachowuje podanej kolejności tak jak np. lista:

moja:

prices = {'Apple': 2.00, 'Banana': 1.00, 'Orange': 1.50, 'Cantaloupe': 4.00, 'Grapes': 0.50}

for key in prices:

print(key, prices[key])

('Orange', 1.5)

('Cantaloupe', 4.0)

('Grapes', 0.5)

('Banana', 1.0)

('Apple', 2.0)

bez nawiasów:

for key in prices:

print key, prices[key]

Orange 1.5 itd.

pix:

prices = {'Apple': 2.00, 'Banana': 1.00, 'Orange': 1.50, 'Cantaloupe': 4.00, 'Grapes': 0.50}

for key, value in prices.items():

print ("%s costs $%s." % (key, value))

Orange costs $1.5.

Cantaloupe costs $4.0.

Grapes costs $0.5.

Banana costs $1.0.

Apple costs $2.0.

----metoda dictionary.setdefault(key,value):

jeśli nie podam wartości to będzie None i już jej za pomocą setdefault nie zmienimy/nadamy

dictionary = {}

dictionary.setdefault("Apple")

print("Before: %s" % dictionary) Before: {'Apple': None}

dictionary.setdefault("Apple", 3)

print("Before: %s" % dictionary) Before: {'Apple': None}

dictionary.setdefault("Apple", 4)

print("After: %s" % dictionary) After: {'Apple': None}

a gdy podamy za pierwszym razem to już zostaje i nie da się metodą setdefault zmienić:

dictionary = {}

dictionary.setdefault("Apple", 3)

print("Before: %s" % dictionary) Before: {'Apple': 3}

dictionary.setdefault("Apple", **4**)

print("After: %s" % dictionary) After: {'Apple': **3**}

---można nowy key dodać z nową wartością:

dictionary = {}

dictionary.setdefault("Apple", 3)

print("Before: %s" % dictionary) Before: {'Apple': 3}

dictionary.setdefault("Jog", 4)

print("After: %s" % dictionary) After: {'Jog': 4, 'Apple': 3}

---

---wersja prosta, czyli wypisze posortowane alfabetycznie imiona czyli po key a nie po wieku:

dict={}

for i in input:

dict.setdefault(i.split('\t')[0],i.split('\t')[1])

for key in sorted(dict):

print key, dict[key]

-----coś się nie sortuje jak trzeba:

input = ['Lekisha 92','Dona 52','Tambra 57','Tyler 7']

dict={}

for i in input:

dict.setdefault(i.split('\t')[0],i.split('\t')[1])

list = sorted(dict.iteritems(), key = lambda x : x[1])

print list [('Dona', '52'), ('Tambra', '57'), ('Tyler', '7'), ('Lekisha', '92')]

for tupla in list:

print "%s\t%s" % (tupla[0], tupla[1])

Dona 52

Tambra 57

Tyler 7

Lekisha 92

---pix tak samo:

dict = {}

for i in input:

name = i.split()[0]

age = int(i.split()[1]) # int tutaj robi całą różnicę!!!

dict.setdefault(name, age)

dict = sorted(dict.iteritems(), key = lambda x : x[1])

for i in dict:

print(i[0]+'\t'+str(i[1]))

---moje poprawione o int teraz działa:

input = ['Lekisha 92','Dona 52','Tambra 57','Tyler 7']

dict={}

for i in input:

dict.setdefault(i.split('\t')[0],**int**(i.split('\t')[1]))

list = sorted(dict.iteritems(), key = lambda x : x[1])

for tupla in list:

print "%s\t%s" % (tupla[0], tupla[1])

Tyler 7

Dona 52

Tambra 57

Lekisha 92

-----------------OBSŁUGA WYJĄTKÓW------------------

try:

print(2 / 0)

except:

print("Exception - division by zero.")

-----------przekonwertuj podane stringi w integery. Zabezpiecz się przed błędami wyjątkami - w przeypadku błedu wypisz 'An exception occurred.'.

41642

0lhp7c

4gcnv

for i in input:

try:

print(int(i))

except:

print('An exception occurred.')

41642

An exception occurred.

An exception occurred.

----różne rodzaje wyjątków - za każdym razem obsługiwany jest tylko jeden:

try:

print(2 / 0)

except ZeroDivisionError:

print("Raised when division or modulo by zero takes place for all numeric types.")

list = ['jeden element']

try:

print(list[1]) # jest tylko index 0, nie ma 1 na tej liście

except IndexError:

print("Raised when an index is not found in a sequence.")

try:

print(x) # nie ma zmiennej x bo jej nie zrobiliśmy

except NameError:

print("Raised when an identifier is not found in the local or global namespace.")

try:

len(2) # nie można zmierzyć len() integera

except TypeError:

print("Raised when an operation or function is attempted that is invalid for the specified data")

---

try:

print(x / 0)

except(NameError, ZeroDivisionError): # tu musi być taka nazwa. Np. "nameErrorek" wywala błąd, jak wywala prawdziwy error, to można tam przeczytać jego nazwę)

print("An exception occured.")

----

91 3

77 2

26 0

for i in input:

try:

x=float(i.split('\t')[0])/float(i.split('\t')[1])

print x

except(ZeroDivisionError):

print ('Division by 0 exception.')

30.3333333333

38.5

Division by 0 exception.

---dodanie else powoduje że wypisze się jeśli nie wystąił żaden wyjątek:

try:

print(2 \*\* 5)

except:

print("An exception occured.")

else:

print("Everything is fine.")

#

32

Everything is fine.

---pożyteczne zwłaszcza gdy nie widać wyniku działania np.:

try:

2 /1

except:

print("An exception occured.")

else:

print("Everything is fine.")

---finally:

try:

print(2 / 0)

except:

print("An exception occured.") # An exception occured.

else:

print("Everything is fine.")

finally:

print("Bye!") # Bye!

---słowo kluczowe assert do zwracania wyjątku "AssertionError" wewnątrz funkcji:

def kelvinToFahrenheit(temperature):

assert temperature >= 0, "Colder than absolute zero!"

return ((temperature - 273) \* 1.8) + 32

print kelvinToFahrenheit(273) # 32.0

print kelvinToFahrenheit(505) # 449.6

print kelvinToFahrenheit(-5) # wywala błąd o nazwie "Colder tha absolute zero!"

----fibbonaci, różne warianty:

input = ["6", "5"]

i = 0

a=0

b=1

while i < int(input[0]):

print (b)

w = a+b

a=b

b=w

i+=1

def fib(x):

i=0

a,b = 0,1

while i<x:

print a

c = a+b

a = b

b = c

i+=1

print

fib(6)