```
# print("Hello \tworld")
#
# print("To jest\n\tnowe")
#
# print(1, 2, 3, 4, sep = " ") #sep = separate

# print("x", "y", "z", sep=" ", end=" takie buty") #end - to co wyświetla się na końcu

# print("sdf", 3456, -2, sep = "\n")

# // , //= dzielenie liczb całkowitych lub iloraz przy dzieleniu z resztą

# print(bool(0))
# print(bool(1))

# Typ logiczny bool to dwie stałe: True (dawniej równe 1) i False (dawniej
# równe 0).

# True i False pojawiły się w Py2.3. Jednak ogólnie w Pythonie jako
# fałsz logiczny traktuje się:

# ▶ liczbę zero (0, 0.0, 0e0, 0j, itp.),
# ▶ stałe False i None (null),
# ▶ puste kolekcje (pusta lista, tuple/krotka, słownik, itp.),
# ▶ puste napisy,
# ▶ obiekty posiadające metodę __nonzero__(), jeśli zwraca ona False lub 0.
# ▶ obiekty posiadające metodę __len__(), jeśli zwraca ona 0

# print(bool('')) #False
# print(bool(' ')) #True

# a = int(input ("Wprowadź pierwszą liczbę : "))
# b = int(input ("Wprowadź drugą liczbę : "))
# print(a/b)

# a = int(input ("Wprowadź pierwszą liczbę : "))
# b = int(input ("Wprowadź drugą liczbę : "))
# if b == 0:
#     print("Błąd")
# else:
#     print(a/b)

# Minimum dwóch liczb:
# a = int(input ("Wprowadź pierwszą liczbę: "))
# b = int(input ("Wprowadź drugą liczbę: "))
# if b > a:
#     mmin = a
# else:
#     mmin = b
# print(mmin)

# Dodatnia, ujemna czy 0?
# a = int(input ("Wprowadź liczbę : "))
# if a > 0:
#     print("a is positive")
# elif a < 0:
#     print("a is negative")
# else:
#     print("a is 0")

# Dni tygodnia:

# a = int(input("Podaj liczbę od 1 do 7 abu sprawdzić dzień tygodnia: "))
# if a == 1:
#     print("Poniedziałek")
```

```
# elif a == 2:
#     print("Wtorek")
# elif a == 3:
#     print("Środa")
# elif a == 4:
#     print("Czwartek")
# elif a == 5:
#     print("Piątek")
# elif a == 6:
#     print("Sobota")
# elif a == 7:
#     print("Niedziela")
# else:
#     print("Wprowadzona liczba nie jest z przedziału od 1 do 7")

# Sprawdzanie czy liczba jest większza od 0 ale mniejsza od 100
# a = int(input("Podaj liczbę: "))
# if a > 0 and a < 100:
#     print("Tak")
# else:
#     print("Nie")

# Jeżeli w if nic nie wykonujemy to musimy tam wpisać pass
# a = 200
# b = 20
# if b > a:
#     pass
# else:
#     print("Hello")

# sprawdzanie
# i = -1
# while i < 6:
#     print(i, end = ', ')
#     i += 1
# else:
#     print("i jest już mniejsze od 6")

# i = 10
# while i < 20:
#     print(i, end = ', ')
#     i += 2
#     if i == 15:
#         break
# else:
#     print("i się skończyło na 20")

# Wyznaczyć taką najmniejszą liczbę naturalną, że suma liczb od niej mniejszych
# i podzielnych przez 7 jest większa od założonej wartości.

# print("podaj liczbę: ")
# varMin = (int(input()))
# suma = 0
# i = 1
# while suma < varMin:
#     if i % 7 == 0:
#         suma += i
#     i += 1
# print("Wyznaczona liczba: ", i - 6)
# print("Suma: ", suma)

# range - Generuje nam ciąg liczb (dedykowany typ range). Trzeba zamienić na listę “by
# podejrzeć”.
# Uwaga : wszystkie argumenty muszą być w typie całkowitym. Jeden argument
# - to “koniec” - ciąg tworzą liczby naturalne od 0 do n – 1. Krok domyślny to 1.
# Dwa argumenty - to “początek” i “koniec”. Krok domyślny to 1. Wtedy
```

```
# wyświetlone są liczby całkowite z przedziału lewostronnie domkniętego .
# Trzy argumenty - to "początek", "koniec" oraz krok.
# print(list(range(5)))#[0, 1, 2, 3, 4]
# print(list(range(1, 11)))#[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# print(list(range(0, 30, 5)))#[0, 5, 10, 15, 20, 25]
# print(list(range(0, 10, 3)))#[0, 3, 6, 9]
# print(list(range(0, -10, -1)))#[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
# print(list(range(0)))#[]
# print(list(range(1, 0)))#[]

# Liczby parzyste od 0 do n:
# Pamiętaj o: początek, koniec, krok

# n = 24
# for x in range(0, n+1, 2):
#     print(x)

# lub

# n = 24
# for x in range(2, n+1):
#     if x % 2 == 0:
#         print(x)

# for x in range(6):
#     if x==3:
#         break
#     print(x)

# for x in range(6):
#     print(x, end = " ")
# else:
#     print("Finished!")

# for i in range(0, 12):
#     if i == 4:
#         continue
#     print(i, end = " ")

# Zakres minimalny i maksymalny liczb oraz suma parzystych liczb w zakresie:

# print("Podaj zakres liczb od - do")
# zmin = int(input("Podaj zakres minimalny: "))
# zmax = int(input("Podaj zakres maximalny: "))
# if zmax < zmin:
#     print("Błąd")
# suma = 0
# for i  in range(zmin, zmax+1):
#     if i % 2 == 0:
#         suma += i
#         i += 1
# print(suma) # jest poprawnie!

# Zagnieżdżone pętle:

# for i in range(4):
#     for j in range(4):
#         print(i, "*", j, "=", i * j)

# Pętla for nie może być pusta, używaj pass.
# for x in [0, 1, 2]:
#     pass
# Uwaga: najczęściej pass jest używany, jeżeli kod będzie dodany później.

# for i in range(3):
#     print(i, end = " ")
```

```
#     for i in range(3):
#         pass

# for i in range(3):
#     for i in range(3):
#         pass
#     print(i, end = " ")

# a = 0
# b = 0
# for i in range(3):
#     for i in range(3):
#         a += 1
#     b += 1
# print('a=', a)
# print('b=', b)

# W 2005 roku posadziłam choinka mierzącą 1,20 m. Przyrasta o 30 cm rocznie.
# Postanowiłam wyciąć gdy przekroczy 7 m. W którym roku będę ścinała
# drzewo?

# year = 2005
# h = 1.20
#
# while h < 7:
#     year += 1
#     h += 0.3
#
# print(year)

# def przywitanie():
#     print ("Pozdrowienia z mojej funckji!")
# przywitanie()

# Funkcja czysta – funkcja, która nie ma żadnych skutków ubocznych, tzn. nie
# modyfikuje ani przekazanych argumentów, ani globalnego stanu programu.

# a = 1
#
# def foo():
#     a = 2
#     print(a)
#
# foo()
# print(a)

# Kolejny przykład

# a = 1
#
# def foo():
#     print(a) #1
# foo() # Wywołanie funkcji.
# print(a) #1

# Funkcja, która nie jest czysta, jest związana ze zmiennymi globalnymi.

# a = 1
# def foo():
#     global a
#     a += 1
#     print (a) #2
# foo() # Wywołanie funkcji.
# print(a) #2

# Funkcja z parametrami
```

```
# def przywitanie_imienne(imie, zyczenia):
#     print ("Witaj " + imie + ". Życzę Tobie " + zyczenia)
# przywitanie_imienne("Jurek", "śmierci")

# def my_sum(a, b):
#     print(a + b)
#
# my_sum(1, 5)

# def my_div(a, b):
#     print(a - b)
#
# my_div(5, 2)

# a = int(input("Podaj pierwszą liczbę: "))
# b = int(input("Podaj drugą liczbę: "))
#
# def mnozenie(a, b):
#     print(a * b)
#
# mnozenie(a, b)

# def my_sum(a, b):
#     a += b
#     print (a)
# a = 3
# my_sum(a, 2)
# print(a)

# def my_sum2(a, b):
#     a += b
#     print(a)
# a = 4
# my_sum2(a,2)
# print(a)

# def my_sum(a, b): #Tutaj jest tak, bo dla wywołanie przypisuję x, a nie a. Mimo
wszystko, sumuje mi x i b.
#     a += b
#     print(a)
# x = 7
# my_sum(x,3)
# print(x)

# def my_sum(a, b):
#     print(a + b)
#
# my_sum("Hello ", "World")
# my_sum(2.21, 3.02)

#Funkcja, która zwraca wartość.

# def przywitanie():
#     return "Siema"
# przywitanie() # Wywołanie funkcji.
# print(przywitanie())
# a = przywitanie()
# print(a)

# def my_sum(a, b):
#     a = + b
#     return a
# x = 2
# y = 3
# print(my_sum(x,y))
```

```
# z = my_sum(x,y)
# print(z)

# W Pythonie domyślnie zwracana jest watrość None

# def my_sum(a, b):
#     a = + b
# print(my_sum(3,5))

# def przywitanie():
#     print ("Witaj")
# przywitanie() # Wywołanie funkcji.
# print(przywitanie())
# a = przywitanie()
# print(a)

# def func():
#     return 5
#     print("Hello")
#     return 3 # Nie działa po pierwszym return
# x = func()
# print(x) #5

# def my_abs(x):
#     if x < 0:
#         return -x
#     else:
#         return x
#     return x
# print(my_abs(-10)) #10
# print(my_abs(10)) #10

# def przywitanie_imienne(imie, zyczenia = "zdrowia" ):
#     print ("Witaj, " + imie + ". Zycze Tobie " + zyczenia)
# przywitanie_imienne("Jacek") # Wywołanie funkcji z domyslnym argumentem
# przywitanie_imienne("Jacek", "szczęścia") # Wywołanie funkcji.
# przywitanie_imienne(imie = "Jacek")

# def suma(a, b=0, c=0, d=0): #Jeśli mam podane inne argumenty za zero, to w wywoływaniu
funkcji mogę je pominąć
#     return a + b + c + d
# print(suma(12, 4)) #16
# print(suma(3, 4, 5, 7)) #19
#
# def prod(a, b=1, c=1, d=1): #Tutaj jedynki bo mnożenie, inaczej by się wyjebało
#     return a * b * c * d
# print(prod(12, 4)) #48
# print(prod(3, 4, 5, 7)) #420

# Funkcje: dwie funkcje z rożną ilością argumentów nie są możliwe
# def area(l, b):
#     return l * b
# def area(r):
#     return 3.14 * r ** 2
# area(3, 4)
# area(7)

# def aFunction(a):
#     if type(a) == str:
#         print('you gave string as argument')
#     if type(a) == int:
#         print('you gave a int as argument')
# aFunction('test')#you gave string as argument
# aFunction(2.0) #nic nie wyświetla bo to jest float a nie int
# aFunction(5)#you gave an int as argument
```

```
# Suma kolejnych liczb:
# def suma(n):
#     s = 0
#     for i in range(1,n):
#         s+=i
#     return s
# print(suma(6)) #15

# Silnia:
# def silnia(n):
#     s = 1
#     for i in range(1,n):
#         s*=i
#     return s
# print(silnia(6)) #120

# def sumaprod(n):
#     s = 0
#     p = 1
#     for i in range(1,n):
#         s+=i
#         p*=i
#     return s, p
# print(sumaprod(6))
# a, b = sumaprod(6)
# print(a)
# print(b)
# a = sumaprod(6)[0] # to w [] to jest pierwszy element krotki
# b = sumaprod(6)[1] # a to jest drugi element krotki
# print(a, ' ', b)

# W wielu językach programowania (jak Java czy C++), nie jest możliwe, aby
# wyrażenia podobne do tych na końcu programu widniały ot tak, samodzielnie.
# Wymagane jest by były one częścią specjalnej funkcji o nazwie main() (funkcja
# główna).
# Pomimo tego, że nie jest to wymagane w języku Python, nie będzie to zły
# pomysł, jeżeli włączymy je w logiczną strukturę programu.

# x = 5
# y = 3
# print(x+y) #8
# def main():
#     x = 5
#     y = 3
# print(x + y)
# main()

# Zanim interpreter języka Python uruchomi Twój program, definiuje on kilka
# specjalnych zmiennych. Jedną z nich jest zmienna __name__, do której
# automatycznie przypisywana jest wartość "__main__"w momencie, gdy dany
# kod uruchamiany jest niezależnie, jako samodzielny program. Z drugiej strony,
# gdy dany kod importowany jest z poziomu innego programu, wartość zmiennej
# __name__ ustawiana jest jako nazwa tego właśnie modułu. Oznacz to, że
# wiemy, czy moduł uruchamiany jest jako niezależny program czy może używany
# jest przez inny program. Bazując na tej wiedzy, możemy zdecydować, czy
# uruchamiać wybraną część kodu.
# Załóżmy przykładowo, że napisaliśmy szereg funkcji, wykonujących proste
# obliczenia matematyczne. Wywołania tych funkcji możemy zamieścić w funkcji
# main. Jest jednak bardziej prawdopodobne, że funkcje te będą zaimportowane
# przez inny program, do zupełnie innych celów. W tym przypadku, nie będziemy
# raczej chcieli wywoływać funkcji main.

# def main():
#     x = 5
#     y = 3
#     print(x + y)
```

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

# W tym kodzie znajdziemy wyrażenie if, aby sprawdzić wartość zmiennej
# __name__. Jeżeli będzie to __main__, wtedy wywołana będzie funkcja main.
# W innym przypadku możemy założyć, że powyższy program został
# zaimportowany z poziomu innego programu, nie będziemy zatem chcieli
# wywołać funkcji main, gdyż ów nowy program będzie używać powyższych
# funkcji wedle potrzeb. Możliwość warunkowego uruchamiania funkcji main
# może się okazać bardzo potrzebna, gdy piszemy program, który potencjalnie
# będzie używany przez innych. Umożliwia nam też dodanie tej funkcjonalności,
# której użytkownik kodu nie będzie potrzebował. Najczęściej będą to procedury
# testowe sprawdzające poprawne funkcjonowanie funkcji.

# Funkcje: nie znana ilość argumentów:

# def suma(a, b):
#     return a + b
# print(suma(12, 4)) #16

# Funkcje: *args

# def suma(a=0, b=0, c=0, d=0):
#     return a + b + c + d
#
# print(suma(12, 4))
# print(suma(3, 4, 5, 7))
# print(suma())

# def suma(*values):
#     s = 0
#     for i in values:
#         s+=i
#     return s
#
# print(suma(12, 4))# 16
# print(suma(3, 4, 5, 8)) #20

# Funkcje wbudowane sum() i len():

# def suma(*values):
#     return sum(values)
# print(suma(12, 4))
# print(suma(3, 4, 5, 7))
# #print(sum(3, 4, 5, 7)) #błąd
# print(sum((3,4,5,7))) #omówimy później

# def srednia(*values):
#    return (sum(values)) / (len(values))
# print(srednia(2, 3, 4, 6)) # 15/4 = 3.75
# print(srednia(45)) # 45/1 = 45.0

# def srednia_wazona(*argumenty):
#     return sum(argumenty) / len(argumenty)
# print(srednia_wazona(2,5,4,6,3,4,6,3,2,5))

# MAŁA DOPISKA, NIE MOŻNA ZROBIĆ SILNI W FUNKCJI *args

# Funkcja długości, napisana samodzielne:

# def dlugosc(*wartosci):
#     i = 0
#     for x in wartosci:
#         i += 1
#     return i
# print(dlugosc(1,2,3,4))
```

```
# Argumenty oprocz *args:

# def func(potega, *values):
#     s = 0
#     for i in values:
#         s+=i**potega
#     return s
# print(func(2, 3, 4, 6)) #61 --> *values to są liczby, do której jest przypisywana
potęga 2.
# print(func(3, 4)) #64
# print(func(3)) #0

# def func(*values, potega):
#     suma = 0
#     for i in values:
#         suma += i**potega
#     return suma
# #print(func(2,3, 5)) --> blad, bo nie jest przypisana potega
# #print(func(potega = 5, 2,3)) blad, bo nie jest przypisane *values
# print(func(3, 4, 5, potega = 3)) # a tutaj jest wszystko git bo *values są podane, a
potem potęga

# Funkcje: dwie funkcje z rożną ilością argumentów nie są możliwe, ale!

# def area(*values):
#     if len(values) == 2:
#         return values[0] * values[1]
#     if len(values) == 1:
#         return 3.14 * values[0] ** 2 # *values[0] bo podaję PIERWSZY argument z LISTY.
Należy pamiętać o kwadratowych nawiasach
# print(area(3, 4)) #12
# print(area(8)) #200.96

# Funkcje: dwie funkcje z rożną ilością argumentów nie są możliwe, ale jeszcze raz

# def area(l = 0, b = 0, r = 0):
#     if r == 0:
#         return l*b
#     else:
#         return 3.14 * r ** 2
# print(area(l=3, b=4)) #12
# print(area(r=7)) #153.86

# lub

# def area(l = None, b = None, r = None):
#     if r == None:
#         return l*b
#    else:
#         return 3.14 * r ** 2
# print(area(l=3, b=4)) #12
# print(area(r=7)) #153.86

# def pole_trojkata(a = None, h = None):
#     if a == None or h == None:
#         return 0
#     else:
#         return (a * h) * 0.5
# print(pole_trojkata(2, 2)) # wszystko git
# print(pole_trojkata(2)) # 0 bo nie mam podanej wartości a lub h

# def p_tr_rownoboczny(a = None):
#     if a == None:
#         return 0
#     else:
```

```
#         return (a ** 2 * 3**(1/2)) * 0.25
# print(p_tr_rownoboczny(4)) # wszystko git
# print(p_tr_rownoboczny()) # nie mam podanej wartości a

# Silnia:

# def silnia(n):
#     if n:
#         return silnia(n-1)*n
#     else:
#         return 1
# print(silnia(5)) #120
# print(silnia(0)) #1
#
# def silnia(n):
#     s = 1
#     for i in range(1,n+1):
#         s*=i
#     return s
# print(silnia(5)) #120
# print(silnia(0)) #1

# Potęgowanie:

#najpierw po mojemu

# def potegowanie(n, p):
#     if p == 0:
#         return 1
#     if p == 1:
#         return n
#     for i in range(1, n+1):
#         prod = n**p
#     return prod
# print(potegowanie(5, 0))
# print(potegowanie(3, 1))
# print(potegowanie(2, 3))

#teraz potegowanie muranovej:

# def pot(n, p):
#     if p == 0:
#         return 1
#     else:
#         return n*pot(n,p-1)
# print(pot(5,2))#25
# print(pot(8,3))#512

# def pot(n, p):
#     prod = 1
#     for i in range (1,p+1):
#         prod *= n
#     return prod
# print(pot(5,2))#25
# print(pot(8,3))#512

# potęgowanie binarne:

# def pot(n, p):
#     if not p:
#         return 1
#     if p%2 == 0:
#         temp = pot(n,p//2)
#         return temp*temp
#     else:
#         temp = pot(n,p//2)
```

```
#     return temp*temp*n
# print(pot(2,10)) #1024

#potęgowanie binarne: Ilość operacji

# global count
# count = 0
# def pot(n, p):
#     global count
#     if not p:
#         return 1
#     if p%2 == 0:
#         temp = pot(n,p//2)
#         count += 1
#         return temp*temp
#     else:
#         temp = pot(n,p//2)
#         count += 2
#         return temp*temp*n
#
# print(pot(2,10))
# print(count) #6
#
# count = 0
# print(pot(2,100)) #duża liczba
# print(count) #10

#potęgowanie zwykłe: Ilość operacji

# count = 0
# def pot(n, p):
#     global count
#     if p == 0:
#         return 1
#     else:
#         count += 1
#         return n*pot(n,p-1)
# print(pot(2,10))
# print(count) #10
#
# count = 0
# print(pot(2,100))
# print(count) #100

# Próba odwzorowania potęgowania zwykłego: Ilości operacji --- udane w 90%

# count = 0
# def potegowanko(n, p):
#     global count
#     if p == 0:
#         return 1
#     else:
#         count += 1
#         return n * potegowanko(n, p-1)
# print(potegowanko(2, 3))
# print(count)

# Ciąg Fibonacciego:

# def fibo(n):
#     if n < 2:
#         return n
#     else:
#         return fibo(n-1) + fibo(n-2) # to jest suma wyrazu ciągu, na którey jest sumą
DWÓCH POPRZEDNICH!!!
# print(fibo(12))
```

```
# Algorytm Euklidesa: Największy wspólny dzielnik:

# def euklides(a, b):
#     if b:
#         return euklides(b, a%b)
#     return a
# print(euklides(12, 18))

# Funkcje wbudowane:

    # abs() - wartosc bezwzgledna
    # min() - minimum
    # max() - maksymum
    # pow() - potega

# Instrukcja pass służy jako symbol zastępczy dla przyszłego kodu, zapobiegając
# błędom z pustych bloków kodu.
# Jest zwykle używana, gdy kod jest zaplanowany, ale jeszcze nie został napisany.

# def future_function():
#     pass
# # this will execute without any action or error
# future_function()
# print(future_function()) #None

# print (abs(-10))
#
# def abs(x):
#     return 5
# print(abs(-20))

# assert - pozwala użyć asercji, aby móc skontrolować działanie programu. Jeśli
# warunek asercji nie zostanie spełniony kompilator zgłosi błąd AssertionError.
# x = "hello"
# #if condition returns True, then nothing happens:
# assert x == "hello"
# #if condition returns False, AssertionError is raised:
# assert x == "goodbye"

# Możemy użyć assert, żeby sprawdzić działanie funkcje:

# assert abs(-10) == 10
# assert abs(10) == 10
# assert abs(0) == 0

# def suma(a, b):
#     return a + b
#
# def main():
#     assert suma(3, 5) == 8
#     assert suma(-2, 1) == -1
#     assert suma('a', 'b') == 'ab'
#     assert suma(0.2,0.3) == 0.5
#     # assert suma (249.99, 22.41) == 272.40
#     # assert print(suma (249.99, 22.41))
#
# if __name__ == "__main__":
#     main()

# Funkcje: import math

# import math
#
# a = 0
# b = math.sin(2*math.pi)
```

```
# print(b)
# print(math.isclose(a,b, rel_tol=1e-09, abs_tol=1e-09))

# import math
#
# print(math.e)
# print(math.exp(1))
# print(math.ceil(math.e))
# print(math.floor(math.e))
# print(math.factorial(10))

# Funkcje: reszta

# import math
#
# print(math.remainder(8,3))#-1.0
# print(math.fmod(8,3))#2.0
# print(8%3)#2
# print(math.remainder(8.5,3))#-0.5
# print(math.fmod(8.5,3))#2.5
# print(8.5%3)#2.5
# print(math.remainder(-1e-100,1e100))#-1e-100
# print(math.fmod(-1e-100,1e100))#-1e-100
# print(-1e-100 % 1e100)#1e+100

# print(len.__doc__) #Return the number of items in a container.
# print(print.__doc__)

# Docstringi języka Python to literały ciągów znaków, które pojawiają się zaraz po
definicji funkcji, metody, klasy lub modułu.

# def square(n):
#     """Take a number n and return the square of n."""
#     return n**2
# print(square.__doc__)

# W Pythonie używamy symbolu hash #, aby napisać komentarz jednowierszowy.
# Ale jeśli nie przypiszemy ciągów znakow (napisow) do żadnej zmiennej, działają one tez
jak komentarze. Na przykład:

# 'napis'
# "napis"
# print(2+3)

# Docstringi języka Python to literały ciągów znaków, które pojawiają się zaraz
# po definicji funkcji, metody, klasy lub modułu.

# "I am a single-line comment"
# '''
# I am a
# multi-line comment!
# '''
# print("Hello World")

# Kiedykolwiek napisy są obecne zaraz po definicji funkcji, modułu, klasy lub metody, są
one skojarzone z obiektem jako ich atrybut __doc__.
# Możemy później użyć tego atrybutu, aby pobrać ten docstring.



# Standardowe konwencje pisania docstringów:
#      Jednowierszowe:
# ▶ Mimo że są jednowierszowe, nadal używamy potrójnych cudzysłowów
# wokół tych docstringów, ponieważ można je później łatwo rozszerzyć.
# ▶ Cudzysłowy zamykające znajdują się w tym samym wierszu co cudzysłowy
```

```
# otwierające.
# ▶ Nie ma pustego wiersza ani przed, ani po docstringu.
# ▶ Nie powinny być opisowe, raczej muszą być zgodne ze strukturą „Zrób to,
# zwróć tamto” kończącą się kropką.

# def multiplier(a, b):
#     """Take two numbers and return their product."""
#     return a*b
# print(multiplier.__doc__)
# print(multiplier(2,90))

# Wielowierszowe cidocstringi składają się z wiersza podsumowującego, tak jak
# jednowierszowe ciągi dokumentacyjne, po którym następuje pusty wiersz, a następnie
bardziej szczegółowy opis.

# Docstringi wielowierszowe dla funkcji:
# ▶ Docstring dla funkcji powinien podsumowywać jej zachowanie i dokumentować jej
argumenty i wartości zwracane.
# ▶ Powinien również zawierać listę wszystkich wyjątków, które mogą zostać zgłoszone,
oraz inne opcjonalne argumenty.

# def add_bin(a, b):
#     '''
#         Return the sum of two decimal numbers in binary digits.
#             Parameters:
#                 a (int): A decimal integer.
#                 b (int): Another decimal integer.
#             Returns:
#                   binary_sum (str): Binary string of the sum of a and b.
#     '''
#     binary_sum = a + b
#     return binary_sum
# print(add_bin.__doc__)
# print(add_bin(3,5))

# Możliwe jest pisanie programów, które obsługują wybrane wyjątki:

# x = 'napis'
# try:
#     print(x + 3)
# except:
#     print("An exception occured")

# #x = 'napis'
# try:
#     print(x + 3)
# except TypeError:
#     print("TypeError")
# except:
#     print("Unknown Error has occured")

# x = 'napis'
# try:
#     print(x + 3)
# except NameError:
#     print('NameError') # Nie działa

# x = 'napis'
# try:
#     print(x + 3)
# except TypeError:
#     print('TypeError') # Działa
#
# x = 'napis'
# try:
```

```
#     print(x + 3)
# except:
#     print("Error") # Działa

# Instrukcja try działa następująco:
# ▶ W pierwszej kolejności wykonywane są instrukcje pod klauzulą try - pomiędzy słowami kluczowymi try i except.

# ▶ Jeżeli nie wystąpi żaden wyjątek, klauzula except jest pomijana i zostaje zakończone wykonywanie instrukcji try.

# ▶ Jeśli wyjątek wystąpi podczas wykonywania klauzuli try, reszta klauzuli
# jest pomijana. Następnie, jeśli jego typ pasuje do wyjątku nazwanego po
# słowie kluczowym except, wykonywana jest klauzula except, a następnie
# wykonanie jest kontynuowane po bloku try/except.

# ▶ Jeśli wystąpi wyjątek, który nie pasuje do wyjątku nazwanego w klauzuli
# except, jest on przekazywany do zewnętrznych instrukcji try; jeśli nie
# zostanie znaleziona obsługa, jest to nieobsłużony wyjątek i wykonanie
# zatrzymuje się z komunikatem błędu.


# Poniższy przykład, prosi użytkownika o wprowadzenie danych, dopóki nie zostanie wprowadzona poprawna liczba całkowita.

# while True:
#     try:
#         print(int(input('Podaj liczbę: ')))
#         break
#     except ValueError:
#         print('Ups! To nie była poprawna liczba. Spróbuj ponownie... (ValueError)')


# ▶ Blok try pozwala przetestować fragment kodu pod kątem błędów.
# ▶ Blok except pozwala obsłużyć błąd.
# ▶ Blok else pozwala wykonać kod, gdy nie wystąpi żaden błąd.
# ▶ Blok finally pozwala wykonać kod, niezależnie od wyniku bloków try i except.

# Przykład:

# Python code to illustrate
# working of try()
# def divide(x, y):
#     try:
#         # Floor Division : Gives only Fractional
#         # Part as Answer
#         result = x // y
#     except ZeroDivisionError:
#         print("Sorry! You are dividing by zero")
#     else:
#         print("Yeah! Your answer is:", result)
#     finally:
#         # this block is always executed
#         # regardless of exception generation.
#         print('This is always executed.')
# # Look at parameters and note the working of Program
# divide(3, 2)
# divide(3, 0)

# def divide(x, y):
#     try:
#         result = x // y #dzielenie z resztą
#     except ZeroDivisionError:
#         return None
#     else:
```

```
#         return result
#     finally:
#         print('This is always executed')
#
# print(divide(3, 2))
# print(divide(3, 0))

# def divide(x, y):
#     try:
#         result = x // y
#     except ZeroDivisionError:
#         return None
#     else:
#         return result
#     print("This is always executed")

# divide(3, 2) #nic sie nie wyswietli
# divide(3, 0) #nic sie nie wyswietli

# Głównym celem bloku finally jest zapewnienie, że określony kod czyszczący
# zostanie wykonany bez względu na to, co się stanie w bloku try. Nawet jeśli nie
# wystąpi żaden wyjątek do obsłużenia, blok finally i tak wykona się po
# zakończeniu bloku try.


# ------------------------------------------------------------------------------------------
-------------------------------
# Zadanie 4 z kolokwium 2 grupa:

# def zaszyfruj_tekst(tekst, przesuniecie):
#     # Inicjalizacja pustego łańcucha na zaszyfrowany tekst
#     zaszyfrowany = ""
#
#     # Iteracja po każdym znaku w tekście
#     for znak in tekst:
#         # Zwiększamy kod ASCII znaku o przesunięcie
#         zaszyfrowany += chr(ord(znak) + przesuniecie)
#
#     # Zwracamy zaszyfrowany tekst
#     return zaszyfrowany
#
#
# # Przykład użycia
# tekst = "Testowy tekst"
# przesuniecie = 3  # O ile zwiększamy kod ASCII
# zaszyfrowany_tekst = zaszyfruj_tekst(tekst, przesuniecie)
#
# print("Oryginalny tekst:", tekst)
# print("Zaszyfrowany tekst:", zaszyfrowany_tekst)
# ------------------------------------------------------------------------------------------
-------------------------------

# Typ string (inaczej po prostu napis)

# Napisy (string)
# ▶ typ sekwencyjny do przechowywania znaków, ale w odróżnieniu od listy
# jest niezmienny
# ▶ w języku Python nie ma oddzielnego typu znakowego apostrofy i
# cudzysłów można stosować zamiennie, ale konsekwentnie
# Inne nazwy: string, napisy, łańcuchy znaków

# a = "Olsztyn"
# print(a)
# print(a[3])
# # a[2] = 'w'
```

```
# a = "Olsztyn"
# b = "Gdańsk"
# print(a + b)
# print(a * 2)
# print(2 * a)

# Specjalne funkcje
# https://www.ascii-code.com/

# # ▶ chr() – zamienia liczbę całkowitą na znak
# print(chr(97)) #a
#
# # ▶ ord() – zamienia znak na liczbę całkowitą odpowiadającą pozycji w tabeli znaków
# print(ord('a')) #97
#
# # ▶ len() – długość napisu
# print(len('ala ma kota')) #11
#
# # ▶ str() – rzutuje argument na napis
# print(str(9.0)*2) #9.09.

# Krojenie stringów

# Uwaga! Pierwszy znak ma indeks 0
# [początek:koniec:krok]
# krok – domyślenie 1

# b = "Hello, World!"
# print(b[2:5]) #od 2 do 5
# print(b[:5]) #od początku (0) do 5 (tutaj jak poniżej jest co jeden ktok, ale nie muszę
go wtedy wpisywać bo domyślnie 1)
# print(b[:5:1]) #od początku (0) do 5, co jeden krok
# print(b[:5:-1]) # od początku, do 5, ale co minus jeden wyraz, czyli jakby od tyłu leci
# print(b[2:]) # od 3 litery, co do końca co jeden krok
# print(b[2::1]) # tak samo jak powyżej
# print(b[2::-1]) # od 3 litery do końca, co minus jeden krok, czuli tak jakby od końca
# print(b[-5:-2]) #od -5 do -2 (nie chce i sie liczyć xdxd)
# print(b[2:9:2]) # od 3 litery do 10, co 2 kroki


# Porządek leksykograficzny: Zależne od ASCII

# print("a" < "A")
# print("n" > "|")
# print("Abc" < "aTw")
# print("0vccx" < "123")
# print("ABC" < "AB")
# print("AB" < "ABC")
# print("ABC" < "abc")
# print("@" < "A")


#Pętla przez string:

# for x in "banana":
#     print(x)


# Sprawdzanie stringów:

# txt = "The best things in life are free!"
# print("free" in txt)
#
# txt2 = "The best things in life are free!"
```

```
# if "free" in txt2:
#     print("Yes, 'free' is in text2!")
#
# txt3 = "The best things in life are free!"
# print("expensive" not in txt3)
#
# txt4 = "The best things in life are free!"
# if "expensive" not in txt4:
#     print("No, 'expensive' is NOT in txt4.")


# Styl, formatowanie napisów:

# ▶ trzy różne konwencje
# ▶ niektóre rzeczy nie działają w każdej wersji 3.x
# ▶ warto zastanowić się czy warto używać tych konstrukcji? czasem może lepiej skorzystać
z funkcji print?

# a = "abc"
# str = "a to %s" % a
# print(str)
#
# b = 4
# c = 5
# str2 = "%d + %d = %d" % (b, c, b + c)
# print(str2)



# Styl - format:

# a = "abc"
# str = "a to {}".format(a)
# print(str)
#
# b = "xdxdxd"
# str2 = "b equals {}".format(b)
# print(str2)
#
# c = 4.5
# d = 5.5
# str3 = "{} + {} equals {}".format(c, d, c + d) #w tych klamrach falowatych podaję po
prostu wyżej wymienione argumenty i tyle.
# print(str3)

# Styl - f-strings:

# f-ciągi pojawiły się w wersji języka Python 3.6 i obecnie są zalecanym
# sposobem formatowania tekstu. Aby utworzyć f-ciąg, należy:
#
# ▶ Wpisać literę f lub F, a zaraz po niej otwierający cudzysłów lub apostrof.
# ▶ Wewnątrz łańcucha umieścić zmienne lub wyrażenia ujęte w nawiasy klamrowe ({}). Ich
wartości zostaną automatycznie zamienione na tekst.

# a = "abc"
# str = (f'a to {a}')
# print(str)

# c = 4.2
# d = 5
# str2 = (f'{c} + {d} = {c + d}')
# print(str2)

# b = 4.2
# c = 5
```

```
# str3 = f"{b:f} + {c:d} = {b+c:e}"
# print(str3)


# print():

# print(object(s), sep=separator, end=end, file=file, flush=flush):

# ▶ sep='separator' Opcjonalnie. Sposób rozdzielenia obiektów, jeśli jest
# ich więcej niż jeden. Wartość domyślna to " ".
# ▶ koniec='koniec' Opcjonalnie. Końcówka. Wartość domyślna to "\n"
# (przesunięcie wiersza)
# ▶ file Opcjonalnie. Obiekt z metodą zapisu. Wartość domyślna to
# sys.stdout
# ▶ flush Opcjonalnie. Wartość logiczna określająca, czy dane wyjściowe są
# opróżniane (True), czy buforowane (False). Wartość domyślna to False.


# print("Hello World")
# print("Hello World", end="|")
# print("Hello World", end="\n")
# print("Hello", "how are you?")
# print("Hello", "how are you?", sep="---")
# print("Hello", "how are you?", sep="\n")
# print("Hello", "how are you?", sep=" | ")


# Wybrane metody klasy string:

# capitalize()
# Zwraca kopię napisu z pierwszym znakiem zmienionym na wielką literę.
# txt = "ala ma kota"
# x = txt.capitalize()
# print(x)#Ala ma kota
# print(txt)#ala ma kota
# print(txt.capitalize())#Ala ma kota


# count(napis, początek, koniec)
# Zwraca ilość nienachodzących na siebie wystąpień napisu napis w zakresie
# [początek:koniec]. Opcjonalne argumenty początek i koniec są
# interpretowane tak samo, jak w operacji wycinania.
# txt = "I love apples, apple are my favorite fruit"
# x = txt.count("apple", 0, 30)
# print(x)#2


# startswith(), endswith():
# Zwraca wynik sprawdzenia, czy napis jest zaczęty/zakończony napisem
# przyrostek. Przy wystąpieniu argumentu początek, sprawdzenie
# rozpoczyna się od tego znaku. Przy wystąpieniu argumentu początek/koniec,
# porównanie zakończy się na tym znaku.
# txt = "hejka naklejka"
# print(txt.startswith("hejka"))
#
# txt2 = "po po ro po po"
# print(txt2.endswith("po"), txt2.startswith("po"))


# expandtabs([numer wielkości, domyślnie 8 znaków])
# Zwraca kopię napisu ze wszystkimi znakami tabulacji zastąpionymi przez
# znaki spacji. Jeśli wielkość nie zostanie podana, przyjmuje się rozmiar
# tabulacji jako 8 znaków.

# txt = "H\te\tl\tl\to"
```

```
# x = txt.expandtabs(3)
# print(x)
# print(txt.expandtabs())
# print(txt)


# find(podnapis, początek, koniec)
# Zwraca najniższy indeks takiego wystąpienia napisu podnapis, aby napis
# był zawarty w wycinku [początek:koniec]. Opcjonalne argumenty
# początek i koniec są interpretowane tak samo, jak w operacji wycinania.
# Zwraca -1, jeśli napis podnapis nie został znaleziony.

# txt = "Hello, welcome to my world."
# y = txt.find("welcome")
# x = txt.find("siema")
# print(y)#7
# print(x)#jeśli nie ma to zwraca -1

# Funkcja find powinna być używana tylko wtedy, gdy chcemy poznać
# pozycję napisu podnapis w danym napisie. Jeżeli chcemy tylko sprawdzić,
# czy napis podnapis występuje w danym napisie, to należy użyć operatora
# in: podnapis in napis

# txt2 = "Hello, welcome to my world."
# z = "welcome" in txt2
# print(z)


# isalnum()
# Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są znakami
# alfanumerycznymi i napis składa się przynajmniej z jednego znaku.

# isalpha()
# Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są literami i napis
# składa się przynajmniej z jednego znaku.

# isdigit()
# Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są cyframi.

# islower()
# Zwraca wynik sprawdzenia, czy wszystkie litery napisu są małymi literami i
# napis zawiera przynajmniej jedną małą literę.

# txt = 'napis100'
# print(txt.isalnum())#True
# print(txt.isalpha())#False
# print(txt.isdigit())#False
# print(txt.islower())#True


# isspace()
# Zwraca wynik sprawdzenia, czy wszystkie znaki napisu są białymi znakami
# i napis składa się przynajmniej z jednego znaku.

# istitle()
# Zwraca wynik sprawdzenia, czy napis ma strukturę tytułu, to znaczy każdy
# wyraz napisu musi zaczynać się wielką literą i składać wyłącznie z małych
# liter lub znaków nieliterowych.

# isupper()
# Zwraca wynik sprawdzenia, czy wszystkie litery napisu są wielkimi literami
# i napis zawiera przynajmniej jedną wielką literę.

# txt = 'This Is A Title Number 1'
# print(txt.isspace())#False
# print(txt.istitle())#True
```

```
# print(txt.isupper())#False


# ljust(szerokość) l jako left
# Zwraca kopię napisu wyrównaną do lewej w napisie o szerokości
# szerokość. Wypełnienie jest uzyskane za pomocą znaków spacji. Jeśli
# szerokość jest mniejsza od len(s), zwracany jest oryginalny napis.

# lstrip(chars) l jako left
# Zwraca kopię napisu z usuniętymi znakami z początku napisu. W
# przypadku, gdy argument chars nie został podany, lub ma wartość None,
# usunięte zostaną białe znaki. Jeżeli argument ten jest podany i nie ma
# wartości None, musi być typu napisowego. Z początku napisu, na rzecz
# którego wywołana została ta metoda, zostaną usunięte znaki wchodzące w
# skład argumentu chars.

# txt = 'Ala ma 3 koty'
# print(txt.ljust(15),'i psa')#Ala ma 3 koty   i psa
# print(txt.ljust(13),'i psa')#Ala ma 3 koty i psa
# print(txt.lstrip('Al'))#a ma 3 koty
# print(txt.lstrip('Ala ma '))#3 koty
# print(txt)#Ala ma 3 koty


# rjust(szerokość) r jako right
# Zwraca kopię napisu wyrównaną do prawej w napisie o szerokości
# szerokość. Wypełnienie jest uzyskane za pomocą znaków spacji. Jeśli
# szerokość jest mniejsza od len(s), zwracany jest oryginalny napis.

# rstrip(chars) r jako right
# Zwraca kopię napisu z usuniętymi znakami z końca napisu. W przypadku,
# gdy argument chars nie został podany, lub ma wartość None, usunięte
# zostaną białe znaki. Jeżeli argument ten jest podany i nie ma wartości
# None, musi być typu napisowego. Z końca napisu, na rzecz którego
# wywołana została ta metoda, zostaną usunięte znaki wchodzące w skład
# argumentu chars.

# txt = 'Ala ma 3 koty'
# print('Napis', txt.rjust(15))#Napis   Ala ma 3 koty
# print('Napis', txt.rjust(13))#Napis Ala ma 3 koty
# print('Napis...', txt.rjust(13))#Napis... Ala ma 3 koty
# print(txt.rstrip(' koty'))#Ala ma 3
# print(txt)#Ala ma 3 koty

# replace(stary, nowy, ile)
# Zwraca kopię napisu z wszystkimi wystąpieniami napisu stary
# zastąpionymi przez nowy. Jeśli zostanie podany argument ile, zostanie
# zastąpiona tylko podana ilość wystąpień.

# rfind(napis , początek, koniec)
# Zwraca najwyższy indeks wystąpienia napisu napis, takiego, aby napis był
# zawarty w przedziale [początek, koniec). Opcjonalne argumenty
# początek i koniec są interpretowane tak samo, jak w operacji wycinania.
# Zwraca -1, jeśli napis nie został znaleziony.

# txt = 'Ala ma 3 kota i jeszcze kota'
# print(txt)#Ala ma 3 kota i jeszcze kota
# print(txt.replace('kota', 'psa',))#Ala ma 3 psa i jeszcze psa
# print(txt)#Ala ma 3 kota i jeszcze kota
# print(txt.rfind('psa'))#-1 bo nie ma w oryginalnym tekście
# print(txt.rfind('kota'))#24 szuka od prawej strony
# print(txt.find('kota'))#9 szuka od lewej storny

# -------------------------------------------------
# print(txt.endswith('kota',24))
# -------------------------------------------------
```

```
# startswith(prefix, start, end)
# Zwraca wynik sprawdzenia, czy napis zaczyna się napisem prefix. Przy
# wystąpieniu argumentu start, sprawdzenie rozpoczyna się od tego znaku.
# Przy wystąpieniu argumentu end, porównanie zakończy się na tym znaku.

# strip(chars)
# Zwraca kopię napisu z usuniętymi znakami z początku i końca napisu. W
# przypadku, gdy argument chars nie został podany, lub ma wartość None,
# usunięte zostaną białe znaki. Jeśli argument ten jest podany i nie ma
# wartości None, musi być typu napisowego. Z początku i końca napisu, na
# rzecz którego wywołana została ta metoda, zostaną usunięte znaki
# wchodzące w skład argumentu chars.

# txt = 'Ala ma 3 kota i jeszcze kota'
# print(txt.startswith('Ala'))#True
# print(txt.startswith('ma',3))#False
# print(txt.startswith('ma',4))#True
# print(txt.startswith('ma',4, 6))#True
# print(txt.strip('Ala kota'))#ma 3 kota i jeszcze
# print(txt)#Ala ma 3 kota i jeszcze kota

# swapcase()
# Zwraca kopię napisu z małymi literami zamienionymi na wielkie, a wielkimi
# na małe.

# title()
# Zwraca kopię napisu zamienioną na strukturę tytułu, to znaczy każdy
# wyraz napisu zostaje zamieniony na rozpoczynający się wielką literą, z
# pozostałymi literami zamienionymi na małe.

# center(szerokość)
# Zwraca kopię napisu wyrównaną po centrum w napisie o szerokości
# szerokość. Wypełnienie jest uzyskane za pomocą znaków spacji. Jeśli
# szerokość jest mniejsza od len(s), zwracany jest oryginalny napis.

# txt = 'Ala ma 3 kota i 2 PSA'
# print(txt.swapcase())#aLA MA 3 KOTA I 2 psa
# print(txt.title())#Ala Ma 3 Kota I 2 Psa
# print('Napis',txt.center(29),'!') #wyśrodkowywuje tekst pomiędzy argumentami
# print('Napis',txt.center(30),'!') #wyśrodkowywuje tekst pomiędzy argumentami (tu zwróć
uwagę na spacje pomiędzy argumentami)
# print('Napis',txt.center(15),'!') #oryginalny napis len(txt) jest większy niż podana
wartość w center
# print(txt.center(12)) # nie ma czego wyśrodkować
# print(txt)#Ala ma 3 kota i 2 PSA


# zfill(szerokość)
# Zwraca napis uzupełniony z lewej strony zerami do podanej szerokości. W
# przypadku, gdy wartość argumentu jest mniejsza od długości napisu,
# zostanie zwrócony oryginalny napis.

# txt = 'Ala'
# num = '1371'
# print('Napis:',txt.rjust(10))#Napis: Ala
# print('Napis:',txt.zfill(10))#Napis: 0000000Ala
# print('Napis:',num.rjust(10))#Napis: 1371
# print('Napis:',num.zfill(10))#Napis: 0000001371
# print('Napis:',num.rjust(5))#Napis: 1371
# print('Napis:',num.zfill(5))#Napis: 0000001371
#
#
# print('Napis:',txt.ljust(10))
# print('Napis:', txt.zfill(10))
```

```
# Stałe zdefiniowane w tym module to:

import string #trzeba zaimportować

# string.ascii_letters
# Konkatenacja stałych ascii_lowercase i ascii_uppercase opisanych
# poniżej. Ta wartość nie zależy od ustawień regionalnych.

# string.ascii_lowercase
# Małe litery alfabetu łacińskiego: 'abcdefghijklmnopqrstuvwxyz'.
# Ta wartość nie zależy od ustawień regionalnych i nie ulega zmianie.

# string.ascii_uppercase
# Wielkie litery alfabetu łacińskiego: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
# Ta wartość nie zależy od ustawień regionalnych i nie ulega zmianie.

# string.digits
# Ciąg znaków: '0123456789'.

# string.hexdigits
# Ciąg znaków: '0123456789abcdefABCDEF'.

# string.octdigits
# Ciąg znaków: '01234567'.

# string.punctuation
# Ciąg znaków ASCII, które są uważane za znaki interpunkcyjne w lokalizacji
# C: "!¨#$%&'()*+,-./:;<=>?@[\]^_`| ".

# string.printable
# Ciąg znaków ASCII, które są uważane za znaki drukowalne. Jest to
# połączenie digits, ascii_letters, punctuation, i whitespace.

# string.whitespace
# Ciąg zawierający wszystkie znaki ASCII uznawane za odstępy. Zawiera
# spację, tabulator, znak nowej linii, powrotu karetki, znak nowej strony i
# tabulator pionowy.

# List (Lista) jest kolekcją uporządkowaną i zmienną. Zezwala na duplikowanych członków.

# Lista w Pythonie to tzw. typ sekwencyjny umożliwia przechowywanie elementów różnych
typów.

# Cechy:
# ▶ zmienny (mutable) - umożliwia przypisanie wartości pojedynczym elementom do zapisu
używamy nawiasów kwadratowych
# ▶ poszczególne elementy rozdzielamy przecinkami
# ▶ każdy element listy ma przyporządkowany indeks
# ▶ elementy listy są numerowane od zera
# ▶ listy są uporządkowane
# ▶ listy są dynamiczne (mogą mieć różną długość)
# ▶ listy mogą być zagnieżdżone

# Uwaga! Listy w języku Python są specyficzną strukturą danych nie zawsze
# dostępną w innych językach programowania. Pojęcie listy w całej
# informatyce"szersze". Wyróżnia się np. listy jednokierunkowe, które nie muszą
# mieć indeksu.

# ▶ Pusta lista:

# a = []
# b = list()

# ▶ Lista z liczbami:
# a = [2, 3, 4.5, 5, -3.2]
```

```
# ▶ Lista mieszana:
# b = [’abcd’, 25+3j, True, 1]

    # Listy: właściwości:

# ▶ Kolejność ma znaczenie:
# a = [1, 2, 3, 4]
# b = [4, 3, 2, 1]
# print(a == b) # False

# ▶ Elementy na liście nie muszą być unikalne
# a = [1, 2, 3, 4, 2]
# b = [1, 2, 3, 4]
# print(a)
# print(a == b)

# ▶ Elementy mogą być różnego typu
# a = [1, 2, 3, '2']
# print(a)

# listy: dostęp do elementów:

# Indeks od zera

# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# print(a[1])
# print(a[4])
# print(a[0])
# # print(a[7]) #nie ma takiego elemntu w liście


# ▶ Ujemny indeks:
# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# print(a[-1])
# print(a[-5])
# print(a[-7])


# Listy: krojenie:

# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# print(a[1:4])
# print(a[0:5])
# print(a[0:6:2]) #od zera do 6 (ostatni) co 2 kroki
# print(a[1:6:2]) #od 1 do 6 (ostatni) co 2 kroki

# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# print(a[::-1]) #lista czyta od końca
# print(a[2:])
# print(a[6:0:-2]) #od ostatniego do początku co -2 kroki (czyli od końca)
# print(a[:]) #cała lista

# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# print(a[::2]) #co 2 kroki cała lista
# print(a[::-2]) # co -2 kroki cała lista (od końca)

# Sprawdzanie długości listy:

# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# print(len(a))

# Implementacja samodzielna długości:
# def dlugosc(lista):
#     x = 0
```

```
#     for i in lista:
#         x += 1
#     return x
# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# print(dlugosc(a))

# Listy: specjalne funkcje:

# ▶ Maksimum i minimum?
# Działa wtedy gdy mamy porządek:
#     ▶ liczby <=
#     ▶ napisy – porządek leksykograficzny

# a = [4,-5,3.4,-11.2] #najmniejsza i największa, normalnie
# print(min(a))
# print(max(a))

# b = ['abc', 'ABCd', 'krt', 'abcd'] #porządek leksykograficzny
# print(min(b))
# print(max(b))


# Listy: modyfikacja:
# a = [4,-5,3.4,-11.2]
# a[2] = 'a'
# print(a)

# a = [4,-5,3.4,-11.2]
# a[2] = ['a','b']
# print(a)

# a = [4,-5,3.4,-11.2]
# a[1:3] = ['a','b'] #od pierwszego indeksu do 3 NIE WLICZAJĄC 3 INDEKSU (w prawdzie,
zamienia mi indeks 1 i 2 na SAMO 'a' i 'b; bez []
# print(a)

# a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
# del(a[2])
# print(a)
#
# del a[::-2]
# print(a)
#
# del a[:] #w taki sposób usuwam wszystkie elementy z listy
# print(a)


# Listy: dodawanie i mnożenie przez liczbę całkowitą:

# a = [4,-5,3.4,-11.2]
# b = ['a','b','c']
# print(a+b)
#
# a = [4,-5,3.4,-11.2]
# print(a*2)
# print(2*a)

# Listy: inne funkcje:

#  list.append(x) – dodaje element na końcu listy.
# Równoważnie a[len(a):] = [x]

# a = [1, 3, 'abc', False]
# a.append(5.3)
# print(a)
```

```
# list.extend(iterable) – dodaje elementy z argumenty na koniec listy.
# Równoważnie: a[len(a):] = iterable

# a = [1, 3, 'abc', False]
# b = [3, -2]
# a.extend(b)
# b.extend(a)
# print(a)
# print(b)

# Różnice?:

# a = [1, 3, 'abc', False]
# b = [3, -2]
# a.append(b)
# print(a) #Tak jest różnica, bo dodaje mi CAŁĄ LISTĘ b do lity a


# list.insert(i, x) – wstawia element x na pozycji i

# a = [1, 3, 'abc', False]
# a.insert(0, 'w') #na pozycji 0, wstawia element 'w'
# print(a)


# list.remove(x) – usuwa element z listy (pierwszy od początku)
# a = [1, 3, 'abc', False]
# a.remove(False)
# print(a)

# b = [3, 4, 5, 3]
# b.remove(3) #Czyta listę po kolei, więc usunie pierwszy element, który widzi
# print(b)


# list.pop() – usuwa i zwraca ostatni element
# list.pop(i) – usuwa i zwraca element na pozycji i

# a = [1, 3, 'abc', False]
# print(a.pop()) # tak jak powiedziane, usuwa i zwraca ostatni element (przez print'a)
# print(a)

# b = [3, -4, 6.2, 7]
# print(b.pop(3)) #usuwa ostatni element z listy, czyli 7
# # print(b.pop(2)) #usuwa '6.2' z listy i też go zwraca (w komentarzu, żeby się nie
popierdolilo)
# print(b)


# list.clear() – usuwa wszystkie elementy z listy.
# Równoważnie: del a[:]

# a = [1, 3, 'abc', False]
# a.clear()
# print(a)


# list.count(x) – zwraca ile razy występuje element x na liście
# a = ['abc','xyz','abc','efg']
# print(a.count('abc'))
# print(a.count(4)) #0 bo 4 nie ma w liście


# list.sort() – sortuje listę (o ile elementy można posortować)
```

```
# a = ['abc','xyz','abc','efg']
# a.sort()
# print(a)


# list.reverse() - odwraca kolejność elementów na liście (nie ma nic związku z
sortowaniem!)
# a = [4, 5, -2, 7.3, 9, -22, 23]
# a.reverse()
# print(a)


# Listy: sortowanie według funkcji
# def myfunc(n):
#     return abs(n - 50)
# thislist = [100, 50, 65, 82, 23]
# thislist.sort(key = myfunc) #BARDZO WAŻNE! jeżeli chcę posortować z funkcją, to muszę
przypisać jej tak jakby klucz, żeby użycie tego było możliwe
# print(thislist)
#
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort() #tu nie daję klucza bo nawet tego do funkcji nie przypisuję
# print(thislist)
#
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort(key = str.lower) #klucz mówi mi tutaj o tym, że sortuję listę po
najmniejszych literach
# print(thislist)


# Listy: sortowanie w odwrotnym porządku:
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort()
# print(thislist)

# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort(reverse = True) #powyżej mam sortowanie zaczynając od uppercase, a potem
lowercase (czyli normalnie) tutaj na odwrót.
# print(thislist)
#
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort(reverse = False) # to samo co w pierwszym przykładzie
# print(thislist)
#
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort(key = str.lower)
# print(thislist)
#
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort(reverse = True, key = str.lower) # Reversuję listę, czyli zamieniam
ostatnie na pierwsze, i sortuje przez małe
# co się niweluje trochę i sortuje mi od wielkich liter. Dosyć dziwne...
# print(thislist)
#
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# thislist.sort(key = str.lower, reverse = True) #to samo co powyżej, I guess nie ma
znaczenia co napiszę pierwsze, a co drugie tutaj
# print(thislist)


# Listy: kopie i przypisanie. WAŻNE
# a = [4, 5, -2, 7.3, 9, -22, 23]
# b = a #kopia listy a
# b[2] = 100 #w liście b znajduje się na 2 indeksie '100'
# print(b)
# print(a) #[4, 5, 100, 7.3, 9, -22, 23], tak się dzieje, bo a = b (b zostało zmienione)
```

```
# list.copy() - tworzy kopię listy
# a = [4, 5, -2, 7.3, 9, -22, 23]
# b = a.copy() #dosłowna kopia i to nie jest to samo co lista a
# b[2] = 100
# print(b) #printuje listę b z zamienionym indeksem 2 z -2 na 100
# print(a) #printuje nie zmienioną listę a


# Listy: pętle:

# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# for x in thislist:
#     print(x) #wypisuje mi normalnie po kolei (po nowych liniach) z listy wszystkie
argumenty

# thislist2 = ["banana", "Orange", "Kiwi", "cherry"] #WAŻNY PRZYKŁAD IMO
# for y in thislist2:
#     print(y[0]) #wypisuje mi JEDYNIE pierwszą literę z każdego wyrazu z listy


# List comprehensions: newlist = [x for x in range(10)]

# #Dłuższa wersja z pętlą for
# squares = []
# for x in range(5):
#     squares.append(x ** 2)
# print(squares)
#
# #krótsza wersja
# squares2 = [x**2 for x in range(5)]
# print(squares2)
#
# #Krótsza wersja, zamiast pętli for
# thislist = ["banana", "Orange", "Kiwi", "cherry"]
# [print(x) for x in thislist]


# List comprehensions z . . . if . . .
# newlist = [expression for item in iterable if condition == True]

# fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
# newlist = [x for x in fruits if x != "apple"] #x dla x w fruits jeżeli x != "apple"
(nie wiem czemu to rozumiem wsm xd)
# print(newlist)

# fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
# newlist = []
# for x in fruits:
#     if x != "apple":
#         newlist.append(x)
# print(newlist) #to samo co powyżej, to jest zrobione jedynie na pętli for


# Listy comprehensions z . . . if . . . : przykłady:
# newlist = [x for x in range(10) if x < 5]
# print(newlist)
#
# fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
# newlist = [x for x in fruits if "a" in x]
# print(newlist)
#
# fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
# newlist = [x.upper() for x in fruits]
# print(newlist)
```

```
#
# fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
# newlist = ['hello' for x in fruits] #zamienia mi wszystkie argumenty z pierwszej listy
na 'hello'
# print(newlist)
#
# fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
# newlist = [x for x in fruits if 'cherry' in x]
# print(newlist)
#
# numbers = [1, -6, 8, 9, 10, 2, 4, 3, 11]
# newlist = [0 if x < 5 else 1 for x in numbers]
# print(newlist)


#Krotki (Tuple):

# krotka = 123, 'abc', True
# krotka2 = (123, 'abc', True)
# print(krotka[2])
# print(krotka[0])
#
# print(tuple(range(5)))
#
# a, b = 2.2341, 5
# print(a)
# print(b)


# Krotki (tuple): właściwości
# Pozwala na duplikaty:
# thistuple = ("apple", "banana", "cherry", "apple", "cherry")
# print(thistuple)

# Pozwala na różne typu elementów:
# tuple1 = ("abc", 34, True, 40, "male")
# print(tuple1)

# Długość:
# thistuple = ("apple", "banana", "cherry")
# print(len(thistuple))
# print(thistuple.count('apple')) #tak dla przypomnienia szybkiego
# print(thistuple.index('cherry'))

# Kilka elementów:
# thistuple = ("apple",)
# print(type(thistuple))


# Krotki (funkcje):
# count() – Zwraca liczbę wystąpień określonej wartości w krotce
# index() – Przeszukuje krotkę pod kątem określonej wartości i zwraca pozycję, w której
została znaleziona.


# Krotki (pętle):
# thistuple = ("apple", "banana", "cherry")
# for x in thistuple:
#     print(len(x), end = '|')
#
# print('\n')
# [print(len(x), end = '|') for x in thistuple]


# Krotki i listy:
# thistuple = ("apple", "banana", "cherry")
```

```
# thistupleaslist = ["apple", "banana", "cherry"]
# print(thistuple)
# print(type(thistuple)) #class 'tuple'
# print(type(thistupleaslist)) #class 'list'


# thislist = list(thistuple)
# print(thislist)
# print(type(thislist))


# thattuple = tuple(thislist)
# print(thattuple)
# print(type(thattuple))

# print([x.upper() for x in thistuple])


# Funkcja zwracająca krotki:

# def minmax(list):
#     min = max = list[0]
#     for i in list:
#         if i < min: min = i
#         if i > max: max = i
#     return min, max

# x = minmax([1,2,34,3])
# print(type(x))
# print(x[1])
# print(x[0])


# Zbiory:
# Cechy:
# ▶ poszczególne elementy rozdzielamy przecinkami
# ▶ nieuporządkowane – nie ma indeksów
# ▶ zbiory są dynamiczne (mogą mieć różną długość)
# ▶ zbiory NIE mogą być zagnieżdżone
# ▶ zbiór jest zmienny, ale jego elementy – nie, tzn. nie można zmieniać
# elementów, można tylko dodawać i usuwać.

# Zbiory - pisownia:
# thisset = {element1, element2, ..., elementN}


# ▶ Pusty zbiór:
# b = set() #To jest pusty zbiór
# print(b)
# print(type(b))
# a = {} #niepoprawnie! To jest pusty słownik
# print(a)
# print(type(a)) #dict


# ▶ Zbiór z liczbami:
# b = {2, 3, 4.5, 5, -3.2}
# print(b)
# print(type(b))
# a = set((2, 3, 4.5, 5, -3.2))
# print(a)
# print(type(a))


# ▶ Zbiór mieszany:
```

```
# c = {'abcd',  25+3j, True, 1} #CZEMU TEJ JEDYNKI MI NIE WYPISUJE TO NIE MAM POJĘCIA XD
=== TRUE = 1, dlatego niweluje sie xddddddddddddddddd
# print(c)
# print(type(c))


# Zbiory: właściwości:

# ▶ Kolejność nie ma znaczenia:
# a = {1, 2, 3, 4}
# b = {4, 3, 2, 1}
# print(a == b) #True


# ▶ Elementy mogą być różnego typu:
# a = {1, 2, 3, '2'}
# print(a)


# ▶ Elementy na liście są unikalne:
# a = {1, 2, 3, 4, 2, 3, 4}
# b = {4, 3, 2, 1}
# print(a)
# print(a == b) #True bo set nie ma indeksów, jak się powtarzają elementy z pierwszego
zbioru i drugiego, to nic xd.


# Zbiory: duplikaty:

# ▶ Duplikowane wartości zostaną zignorowane:
# thisset = {"apple", "banana", "cherry", "apple"}
# print(thisset)


# ▶ Wartości True i 1 są uważane za tę samą wartość i traktowane jako duplikaty
# thisset = {"apple", "banana", "cherry", True, 1, 2} # 1 i True to to samo
# print(thisset)


# ▶ Wartości False i 0 są uważane za tę samą wartość i traktowane jako duplikaty
# thisset = {"apple", "banana", "cherry", False, True, 0} # 0 i False to to samo
# print(thisset)


# ▶ Wartości None i td. NIE są uważane za tę samą wartość co False
# thisset = {"apple", "banana", "cherry", False, True, '', None}
# print(thisset)


# Zbiory: co może być elementem?
# Obiekty, które są hashable. Obiekt jest hashable, jeśli ma wartość hash, która
# nigdy się nie zmienia w trakcie jego życia (wymaga metody __hash__()) i może
# być porównywany z innymi obiektami (wymaga metody __eq__()). Obiekty
# hashable, które są porównywane jako równe, muszą mieć tę samą wartość hash.

# x = hash(set((1,2))) #zbior unhashable
# x = hash(frozenset([1,2])) #frozenset to zbiór, którego nie można zmienić, HASHABLE
#x = hash(([1,2], [2,3])) #krotka zmiennych obiektow, unhashable
# x = hash((1,2,3)) #krotka niezmiennych objektow, HASHABLE
#x = hash()
# x = hash([[1,2], [2,3]]) #list zmiennych objektow, unhashable
#x = hash([1,2,3]) #list niezmiennych objektow, unhashable

# Zbiory: co może być elementem, przykłady:
# set1 = {"apple", "banana", "cherry", True, 1, 2}
```

```
# print(set1)

# set2 = {(1,2,3),'a'}
# print(set2)

# set3 = {1,2,3,{1,2}} #UNHASHABLE
# set3 = {1,2,3,frozenset([1,2])}
# print(set3)

#set4 = {1,2,3,[1,2]} #UNHASHABLE


# Zbiory: funkcje:

#maksimum i minimum:
# Działa wtedy gdy mamy porządek:
# ▶ liczby <=
# ▶ napisy - porządek leksykograficzny

# a = set((4,-5,3.7,-11.2))
# print(min(a))
# print(max(a))

# b = set(('abc', 'ABCd', 'krt', 'abcd'))
# print(min(b))
# print(max(b))


# długość:
# a = set((4,-5,3.4,-11.2))
# print(len(a)) #4
# b = set(('abc', 'abc', 'krt', 'abc')) #2
# print(len(b)) #Uwaga!


# Zbiory: dostęp do wartości:

# Nie można używać indeksów!
# ▶ Pętla:
# thisset = {"apple", "banana", "cherry"}
#
# for x in thisset:
#     print(x)

# ▶ sprawdzenie czy jest:
# thisset = {"apple", "banana", "cherry"}
# print('banana' in thisset)

# ▶ sprawdzenie czy nie jest:
# thisset = {"apple", "banana", "cherry"}
# print('banana' not in thisset)


# Zbiory: dodawanie elementów do zbioru:
# ▶ add() - dodaje jeden element
# thisset = {"apple", "banana", "cherry"}
# thisset.add('orange')
# print(thisset)

# ▶ update() - dodaje kilka elementów:
# thisset = {"apple", "banana", "cherry"}
# tropical = {"pineapple", "mango", "papaya"}
# thisset.update(tropical)
# print(thisset)
```

```
# ▶ Obiekt w metodzie update() nie musi być zbiorem, może to być dowolny
# obiekt iterowalny (krotki, listy, słowniki itp.).
# thisset = {"apple", "banana", "cherry"}
# mylist = ["kiwi", "orange"]
# thisset.update(mylist)
# print(thisset)

# a = set((1,2,3))
# a.update(range(1,11))
# print(a)


# Jeszcze update() - UWAGA
# a = {4,3,24}
# a.update('słówko')
# print(a)
#
# a.update(('słówko',)) #Przecinek po 'słówko' jest istotny
# print(a)#{4, 5, 'o', 'slowo', 'l', 23, 'w', 's'}


# Zbiory: usunięcie elementów:
# ▶ remove() - usuwa element:
# thisset = {"apple", "banana", "cherry"}
# thisset.remove("banana")
# print(thisset)

# Jeśli element do usunięcia nie istnieje, remove() zgłosi błąd:
#thisset.remove('orange')
#print(thisset)


# ▶ discard() - usuwa element:
# thisset = {"apple", "banana", "cherry"}
# thisset.discard("banana")
# print(thisset)

# Jeśli element do usunięcia nie istnieje, discard() NIE zgłosi błąd.
# thisset.discard('orange')
# print(thisset)


# ▶ Możesz również użyć metody pop(), aby usunąć element, ale ta metoda
# usunie losowy element, więc nie możesz być pewien, który element
# zostanie usunięty.
# Wartością zwracaną przez metodę pop() jest usunięty element:

# thisset = {"apple", "banana", "cherry"}
# x = thisset.pop()
# print(x)
# print(thisset)


# Zbiory: czyszczenie zbioru:

# ▶ Metoda clear() opróżnia zbiór:
# thisset = {"apple", "banana", "cherry"}
# thisset.clear()

# ▶ Metoda del() usuwa zbiór (jak i inne zmienny):
# thisset = {"apple", "banana", "cherry"}
# del thisset
# #print(thisset)

# a = 5
```

```
# del a
# print(a)


# Zbiory: inne funkcje:
# Istnieje kilka sposobów łączenia dwóch lub więcej zbiorów w Pythonie:

# ▶ Metody union() łączą wszystkie elementy z obu zbiorów.
# ▶ Metoda intersection() - zachowuje TYLKO duplikaty.
# ▶ Metoda difference() zachowuje elementy z pierwszego zbioru, których nie ma w
pozostałych zestawach.
# ▶ Metoda symmetric_difference() zachowuje wszystkie elementy OPRÓCZ duplikatów.

# union: dwa zbiory:
# set1 = {1, 2, 3}
# set2 = {2, 3, 4}
# set3 = set1.union(set2) #nie mogę zrobić po prostu set1.union(set2) niestety
# print(set3)

# ▶ Równoważnie:
# set1 = {1, 2, 3}
# set2 = {2, 3, 4}
# set3 = set1 | set2 # '|' działa jak union, co jest spoko
# print(set1)
# print(set2)
# print(set3)

# union: kilka zbiorów:
# set1 = {1, 2, 3}
# set2 = {2, 3, 4}
# set3 = {"pineapple", "banana"}
# set4 = {"apple", "banana", "cherry"}
#
# bigset = set1.union(set2, set3, set4)
# print(bigset)

# Równoważnie:
# set1 = {1, 2, 3}
# set2 = {2, 3, 4}
# set3 = {"pineapple", "banana"}
# set4 = {"apple", "banana", "cherry"}
# largoseto = set1 | set2 | set3 |set4
# print(largoseto)


# jeszcze o union
# Metoda union() pozwala na połączenie zbioru z innymi typami danych, takimi jak listy
lub krotki.

# Wynikiem będzie zbiór!
# x = {"a", "b", "c"}
# y = (1, 2, 3)
#
# z = x.union(y) #tak jakby dodawanie zbiorów
# print(z)

# Uwaga! Operator | pozwala jedynie na łączenie zbiorów z innymi zbiorami, a nie
# z innymi typami danych, tak jak można to zrobić za pomocą metody union().

# x = {"a", "b", "c"}
# y = {1, 2, 3}
# x|=y# to samo co update
# print(x)
```

```
# Uwaga 1:
# Przypominam ze True jest to samo co 1, a False - to samo, co 0.

# set1 = {"apple", 1, "banana", 0, "cherry"}
# set2 = {False, "google", "apple", 2, True}
#
# set3 = set1.union(set2)
# print(set3) #{0, 1, 2, 'google', 'banana', 'cherry', 'apple'}
#
# set4 = set2.union(set1)
# print(set4)


# set1 = {"apple", 1, "banana", 0, "cherry"}
# set2 = {False, "google", "apple", 2, True}
#
# set3 = set1.union(set2)
# print(set3)
# for x in set3:
#     print(x, ":", type(x))
#
# set4 = set2.union(set1)
# print(set4)
# for x in set4:
#     print(x, ":", type(x))


# intersection - przykłady:

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
# set3 = set1.intersection(set2)
# print(set3)


# set1 = {1, 2, 3}
# set2 = {2, 3, 4}
# set3 = {"pineapple", "banana"}
# set4 = {"apple", "banana", "cherry"}
#
# myset = set1.intersection(set2, set3, set4)
# print(myset) #pusty zbiór, wszystkie zbiory nie mają żadnej wspólnego argumentu


# intersection &
# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
# set3 = set1 & set2
# print(set3)

# set1 = {1, 2, 3}
# set2 = {2, 3, 4}
# set3 = {"pineapple", "banana"}
# set4 = {"apple", "banana", "cherry"}
# myset = set1&set2&set3&set4
# print(myset)

# Uwaga! Operator & pozwala jedynie na łączenie zbiorów z innymi
# zbiorami, a nie z innymi typami danych, tak jak można to zrobić za
# pomocą metody intersection().

# set1 = {1, 2, 3}
# #myset = set1.intersection(2,3,4,6) #błąd
# myset = set1.intersection((2,3,4,6)) #tuple
# print(myset)
```

```
# intersection_update (Remove the items that is not present in both set1 and set2):
# Metoda intersection_update() również zachowa TYLKO duplikaty, ale
# zmieni oryginalny zbiór zamiast zwrócić nowy zbiór.

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
# set3 = set1.intersection(set2)
# print(set1)
# print(set3)
# set4 = set1.intersection_update(set2)
# print(set1)
# print(set4)#None

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
# set1.intersection_update(set2)
# #set1&=set2#to samo
# print(set1)

# Uwaga!:
# Przypominam ze True jest to samo co 1, a False - to samo, co 0.

# set1 = {"apple", 1, "banana", 0, "cherry"}
# set2 = {False, "google", "apple", 2, True}
# set3 = set1.intersection(set2)
# print(set3) #{False, 'apple', True}
# set4 = set2.intersection(set1)
# print(set4) #{0, 1, 'apple'}

# set1 = {"apple", 1, "banana", 0, "cherry"}
# set2 = {False, "google", "apple", 2, True}
# set1.intersection_update(set2)
# print(set1)#{False, True, 'apple'}

# set1 = {"apple", 1, "banana", 0, "cherry"}
# set2 = {False, "google", "apple", 2, True}
# set2.intersection_update(set1)
# print(set2)#{0, 1, 'apple'}


# set1 = {"apple", 1, "banana", 0, "cherry"}
# set2 = {False, "google", "apple", 2, True}
#
# set3 = set1.intersection(set2)
# print(set3)#{False, True, 'apple'}
# for x in set3:
#     print(x, ':', type(x))
# set4 = set2.intersection(set1)
# print(set4)#{0, 1, 'apple'}
# for x in set4:
#     print(x, ':', type(x))


# difference:
# Metoda difference() zwróci nowy zbiór zawierający tylko elementy z pierwszego
# zbioru, których nie ma w drugim zbiorze.

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
#
# set3 = set1.difference(set2)
# print(set3)

# Można używać -
# set1 = {"apple", "banana", "cherry"}
```

```
# set2 = {"google", "microsoft", "apple"}
#
# set3 = set1 - set2
# print(set3) #{'cherry', 'banana'}

# Operator - działa jedynie na dwóch zbiorach, a nie z innymi typami danych, jak
# to możliwe w przypadku metody difference()


# Uwaga na porządek odejmowania!
# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
# set3 = {"banana", "two bananas"}
#
# set4 = set1 - set2 -set3
# print(set4)
#
# set5 = set1 - (set2 - set3)
# print(set5) #{'banana', 'cherry'}


# difference_update:
# Metoda difference_update() również zachowa elementy z pierwszego zbioru,
# których nie ma w drugim zbiorze, ale zmieni oryginalny zbiór zamiast zwrócić nowy.

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
#
# set1.difference_update(set2) #set1-=set2
# print(set1) #{'cherry', 'banana'}


# symmetric_difference():
# Metoda symmetric_difference() metoda zachowa tylko te elementy, które
# NIE są obecne w obu zbiorach.

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
#
# set3 = set1.symmetric_difference(set2)
# print(set3) #{'google', 'banana', 'microsoft', 'cherry'}


# Możesz użyć operatora ^ zamiast metody symmetric_difference(), a otrzymasz ten sam
wynik.

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
#
# set3 = set1 ^ set2
# print(set3) #{'google', 'banana', 'microsoft', 'cherry'}

# Operator ^ działa jedynie na dwóch zbiorach, a nie z innymi typami danych, jak
# to możliwe w przypadku metody symmetric_difference().


# symmetric_difference_update:
# Metoda symmetric_difference_update() również zachowa wszystkie
# elementy oprócz duplikatów, ale zmieni oryginalny zbiór zamiast zwrócić nowy.

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
# set3 = set1.symmetric_difference_update(set2)
# print(set1)#{'cherry', 'microsoft', 'banana', 'google'}
# print(set3)#None
```

```
# To samo co:
# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
# set1^=set2
# print(set1)#{'cherry', 'microsoft', 'banana', 'google'}


# Jeszcze metody:
# isdisjoint() - zwraca informację, czy dwa zbiory mają przecięcie, czy nie
# Return True if no items in set x is present in set y:

# set1 = {"apple", "banana", "cherry"}
# set2 = {"google", "microsoft", "apple"}
#
# print(set1.isdisjoint(set2)) #False
#
# x = {"apple", "banana", "cherry"}
# y = {"google", "microsoft", "facebook"}
#
# z = x.isdisjoint(y)
#
# print(z) #True


# issubset() - Return True if all items in set x are present in set y:
# x = {"a", "b", "c"}
# y = {"f", "e", "d", "c", "b", "a"}
#
# print(x.issubset(y))
#
# set1 = {"apple", "banana", "cherry"}
# set2 = {"apple"}
# print(set1.issubset(set2))#False
# print(set2.issubset(set1))#True


#issuperset() - Return True if all items set y are present in set x:
# x = {"f", "e", "d", "c", "b", "a"}
# y = {"a", "b", "c"}
#
# print(x.issuperset(y))
#
# set1 = {"apple", "banana", "cherry"}
# set2 = {"apple", "banana", "cherry", "microsoft"}
#
# print(set1.issuperset(set2))
# print(set2.issuperset(set1))


# copy - WAŻNE!!!!!!!!!!
# a = {4,5,23}
# b = a
# b.update({100})
# print(b)
# print(a)
#
# c = a.copy()
# c.update(('apple', 'banana'))
# print(c)


# Słowniki (dictionary):
# Jednym z wbudowanych typów w Pytona są tzw. słowniki (ang. dictionary).
# Słownik jest strukturą danych podobną do list z tą różnicą, że słowniki nie
# pracują w oparciu o indeksy, ale w oparciu o parę: klucz - wartość.
```

```
# Słownik to kolekcja, która jest uporządkowany*, zmienna i nie dopuszcza
# duplikatów.

# car = { "brand": "Ford", "model": "Mustang", "year": 1964}
# print(car)

# Możemy też utworzyć pusty słownik i dodawać do niego kolejne elementy:

# car = {} # utworzenie pustego słownika
# print(car)

#dodanie pary klucz-wartość do słownika
# car["brand"] = "Ford"
# car["model"] = "Mustang"
# car["year"] = 1964
# print(car) #{ "brand": "Ford", "model": "Mustang", "year": 1964}


# Słowniki mają tę zaletę, że ich wartości mogą zawierać dowolny typ danych
# (np. napisy, liczby, listy etc.). Z kluczami jest już inaczej, bo muszą być one
# zestawami tego samego typu elementów, np. napisy, liczby etc. Nie da się jako
# zestaw kluczy podać jednocześnie np. listę i liczby – Python zwróci wtedy błąd. TO
WAŻNE JAK COŚ

# Zwracanie do wartości:
# Wartości słownika wywołujemy przez odwołanie się do jego klucza:

# car = { "brand": "Ford", "model": "Mustang", "year": 1964}
# print(car["brand"]) #Słowniki można wywoływać tylko po kluczu
#print(car[0]) #Uwaga - blad!

# car = { "brand": "Ford", "model": "Mustang", "year": 1964}
# car["year"] = 1980
# print(car)
#
# car["price"] = 10000
# print(car)

# Uwaga: łatwo zrobić błąd

# Uwaga! Odwoływanie się przez klucz słownika jest jednoznaczne, ponieważ w
# danym słowniku nie może być dwóch takich samych nazw kluczy. Pamiętaj, że
# nazwy kluczy w słowniku są wrażliwe na wielkość liter! Pamiętaj też, że
# przypisanie wartości do istniejącego już klucza automatycznie nadpisuje starą
# wartość.

# car = { "brand": "Ford", "model": "Mustang", "year": 1964}
# car["Model"] = "Boroco"
# print(car) #{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'Model': 'Boroco'}
# car["model"] = "Boroco"
# print(car) #{'brand': 'Ford', 'model': 'Boroco', 'year': 1964, 'Model': 'Boroco'}


# Inne działania na słownikach:
# Wartości słownika można również poddawać działaniom.
# dict1 = {'key1': 'napis', 'key2': 123, 'key3': ['i1', 'i2', 'i3'], 'key4': [11, 22,
33]}
# print(dict1)

# Modyfikacja elementów słownika:
# dict1['key2'] = dict1['key2'] - 100 #odejmuje mi od 123 stówkę. Dlatego to  jest
napisane, że podjemuje działaniem xd
# print(dict1)

# Usuwanie elementu słownika:
```

```
# dict1 = {'key1': 'napis', 'key2': 123, 'key3': ['i1', 'i2', 'i3'], 'key4': [11, 22,
33]}
# del dict1['key3']
# print(dict1)
#
# # wywołanie i usunięcie elementu słownika za pomocą 'pop'
# a = dict1.pop('key4') #NIE WSTAWIAM KWADRATOWYCH NAWIASÓW W POP-IE
# print(a)
# print(dict1)
#
# b = dict1.popitem()
# print(b) #('key2', 123)
# #do 3.7 usuwa i zwraca losowy element, dalej - ostatni dodany
# print(dict1) #{'key1': 'napis'}


# clear():
# czyści cały słownik:
# dict1.clear()
# print(dict1) #{}

# update:
# Aby zaktualizować słownik w oparciu o inny słownik używamy metody update:

# dict1 = {'k1': 'w1', 'k2': 'w2'} #zdefiniowanie pierwszego słownika
# dict2 = {'k10': 'w10'} #utworzenie drugiego słownika
# dict1.update(dict2)
# print(dict1) #{'k1': 'w1', 'k2': 'w2', 'k10': 'w10'}


# fromkeys():
# fromkeys() zwraca słownik z podanymi kluczami:
# x = ('key1', 'key2', 'key3')
# y = 0
#
# thisdict = dict.fromkeys(x, y)
# print(thisdict) #{'key1': 0, 'key2': 0, 'key3': 0}
#
# x = ('key1', 'key2', 'key3')
# thisdict2 = dict.fromkeys(x)
# print(thisdict2) #0 to w słowniku none, a nie false (tylko w zbiorach jest inaczej)


# get():
# get() wartość według klucza     (Get the value of the "model" item:)
# car = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# } #w taki sposób też można pisać słowniki, ale imo lepiej w ciągu
# x = car.get("model")
# print(x) #Mustang

# To samo co:
# car = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# }
# x = car["model"]
# print(x)#Mustang

# car["model"] mozna zmieniac, car.get("model") = 'abrakadabra' wywołuje błąd


# setdefault():
```

```
# Zwraca wartosc po podanym kluczy, jeżeli nie istnieje - dodaje
# car = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# }
# x = car.setdefault("model", "Bronco")
# print(x) #Mustang

# car = {
# "brand": "Ford",
# "year": 1964
# }
# x = car.setdefault("model", "Bronco") #Tutaj modelu nie ma, więc go dopisuje i ustawia
na defaultowy
# print(x) #Bronco
# print(car) #{'brand': 'Ford', 'year': 1964, 'model': 'Bronco'}
#
# y = car.setdefault("model", "NIE WIEM") #Nie zmienia wartości klucza, bo 'Bronco' jest
już ustawione przez default
# print(y) #Bronco
# print(car) #{'brand': 'Ford', 'year': 1964, 'model': 'Bronco'}


# setdefault() vs getitem()
# car = {
# "brand": "Ford",
# "year": 1964
# }
# x = car.getitem('model') #Błąd
# print(car['model']) #Błąd
# x = car.setdefault("model")
# print(x) #None
# print(car)


# Słowniki: inne metody
# Aby ustalić ile par „klucz - wartość” zawiera słownik, używamy znanej już
# funkcji len. Za pomocą metod items(), keys() lub values() możemy wyłuszczyć
# zawartość słownika.

# Przykład:
# dict1 = {'k1': 'w1', 'k2': 'w2'}
#
# print(len(dict1)) # Sprawdzanie liczby kluczy w słowniku
#
# # Zwracanie (prawie) listy elementów słownika
# print(dict1.items()) # klucz - wartość
# print(dict1.keys()) # kluczy
# print(dict1.values()) # wartości


# Zagnieżdżanie słowników:
# Jak sama nazwa wskazuje słowniki można zagnieżdżać nie tylko innymi listami
# etc., ale również samymi słownikami.

# Definicja słownika zagnieżdżonego
# slownik_zagniezdzony = {'klucz1':{'pod_klucz':{'pod_pod_klucz': 'wartosc1'}}}
# # Wywołanie wartości zagnieżdżonego słownika
# a = slownik_zagniezdzony['klucz1']['pod_klucz']['pod_pod_klucz']
# print(a)

#przykład:

# myfamily = {'child1': {"name" : "Emil", "year" : 2004}, "child2" : {"name" : "Tobias",
"year" : 2007},
```

```
# "child3" : {"name" : "Linus", "year" : 2011}}
#
# print(myfamily["child2"]["name"]) #Tobias


# Pętle: klucze:
# thisdict = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# }

# for x in thisdict.keys():
#     print(x) #brand '\n' model '\n' year

# To samo co na górze
# for x in thisdict:
#     print(x)


# Pętle: wartości:
# thisdict = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# }
#
# for x in thisdict.values():
#     print(x) # Ford '\n' Mustang '\n' 1964

# To samo co na górze:
# for x in thisdict:
#     print(thisdict[x])


# Pętle: klucze i wartości:
# thisdict = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# }
#
# for x, y in thisdict.items():
#     print(x, y) # brand Ford '\n' model Mustang '\n' year 1964

# myfamily = {
#     "child1" : {
#         "name" : "Emil",
#         "year" : 2004
#     },
#     "child2" : {
#         "name" : "Tobias",
#         "year" : 2007
#     },
#     "child3" : {
#         "name" : "Linus",
#         "year" : 2011
#     }
# }
#
# for x, obj in myfamily.items():
#     print(x)
#     for j in obj:
#         print(j + ':', obj[j])

#obj to jest po prostu tak jakby wartość podklucza
```

```
# copy() WAŻNE!!!!

# car = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# }
#
# x = car
# x["model"] = "Boroco"
# print(x)
# print(car)


# car = {
# "brand": "Ford",
# "model": "Mustang",
# "year": 1964
# }
#
# x = car.copy()
# x["model"] = "Boroco"
# print(x) #tutaj zamieniłem model na Boroco
# print(car) #wyświetla mi normalnie słownik


# Funkcja zip():
# Podstawy: funkcja zip łączy ze sobą elementy z różnych obiektów iterowalnych,
# takich jak listy, krotki, zbiory, i zwraca nam iterator. Możemy jej użyć to
# połączenia ze sobą dwóch list

# id = [1,2,3,4]
# leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
#
# record = zip(id, leaders)
# print(record) #<zip object at 0x000002C9825A8300>
#
# print(list(record)) #[(1, 'Elon Mask'), (2, 'Tim Cook'), (3, 'Bill Gates'), (4, 'Yang
Zhou')]

# Jak widać powyżej, funkcja zip zwraca iterator z krotkami, gdzie n-ta krotka
# zawiera n-ty element z każdej z list

# Tak naprawdę to funkcja zip ma w Pythonie o wiele większe możliwości od
# normalnego suwaka - może ona działać z dowolną liczbą obiektów iterowalnych,
# a nie tylko z dwoma. Jeśli przekażemy do funkcji zip listę:

# id = [1,2,3,4]
# record = zip(id)
# print(list(record)) #list jako funkcja jak cos       [(1,), (2,), (3,), (4,)]


# id = [1,2,3,4]
# leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
# sex = ["male", "male", "male", "male"]
# record = zip(id, leaders, sex)
# print(list(record))


# Prawdziwe dane nie zawsze są czyste i pełne - czasami musimy uporać się z
# nierówną długością obiektów iterowalnych. Wynik funkcji zip jest domyślnie
# oparty na najkrótszym z obiektów iterowalnych.

# id = [1, 2]
```

```
# leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
# record = zip(id, leaders)
# print(list(record))


# A co by było, gdybyśmy otrzymali listę record z poprzedniego przykładu i
# chcielibyśmy rozpakować ją do osobnych list? Niestety Python nie ma
# przeznaczonej do tego funkcji. Chociaż, jeśli znamy specjalne użycia *, to
# rozpakowywanie stanie się niezwykle proste.

# record =  [(1, 'Elon Mask'), (2, 'Tim Cook'), (3, 'Bill Gates'), (4, 'Yang Zhou')]
# id, leaders = zip(*record)
# print(list(id))
# print(list(leaders))
# # Można też bez listy
# print(id)
# print(leaders)


# Dzięki funkcji zip, tworzenie i aktualizowanie dict opartego na oddzielnych
# listach jest dosyć proste. Mamy tutaj dwa jednolinijkowe rozwiązania:
# ▶ Używanie wyrażeń słownikowych razem z zip
# ▶ Używanie funkcji dict razem z zip

# ▶ Używanie wyrażeń słownikowych razem z zip
# id = [1,2,3,4]
# leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
# # create dict by dict comprehension
# leader_dict = {i: name for i, name in zip(id, leaders)}
# print(leader_dict)

# ▶ Używanie funkcji dict razem z zip
# create dict by dict comprehension
# leader_dict2 = dict(zip(id, leaders))
# print(leader_dict2)


# Funkcja zip() – dodawanie elementów do słownika
# leaders_dict = leader_dict = {1: 'Elon Mask', 2: 'Tim Cook', 3: 'Bill Gates', 4: 'Yang
Zhou'}
#
# # update
# other_id = [5,6]
# other_leaders = ['Katy Perry', 'Mao Zedong']
#
# leaders_dict.update(zip(other_id, other_leaders))
# print(leader_dict)


# Korzystanie z funkcji zip w pętlach for:

# Często zdarza się, że używamy wielu obiektów iterowalnych na raz. Funkcja zip
# ma tutaj spore pole do popisu, jeśli będziemy jej używać z pętlami for.
# Sprawdźmy, jak to wygląda:

# products = ["cherry", "strawberry", "banana"]
# price = [2.5, 3, 5]
# cost = [1, 1.5, 2]
#
# for produkt, cena, koszt in zip(products, price, cost):
#     print(f'The profit of a box of {products} is {cena - koszt} USD.')
```

```
# Czytanie z plików:

# Proces czytania i zapisu danych do pliku to zadanie złożone. Python jak
# większość języków programowania pozwala wczytywać dane ze zbiorów
# zewnętrznych, jak i zapisywać dane do plików. Proces czytania można
# zrealizowaćć przy pomocy funkcji wbudowanych -open(), read() i close()
# lub – w przypadku danych o ustalonej strukturze – najczęściej tabelarycznej,
# można skorzystać z gotowych funkcji wspierających ten proces.

# Aby odczytać plik tekstowy należy wykonać trzy polecenia:
# ▶ open() aby nawiązać połączenie z plikiem (nic nie zostaje wczytane)
# ▶ read() aby wczytać całą zawartość pliku do jednej zmiennej jako tekst
# ▶ close() zamknąć plik po zakończeniu czytania

# Dodatkowo proces czytania lub zapisu wymaga znalezienia właściwego pliku,
# lub - w przypadku odczytu utworzenia nowego pliku.
# Dalej używa się plik.txt z zawartością

# Wczytanie niewielkiego pliku tekstowego znajdującego się w tym samym
# katalogu co nasz skrypt:

# f = open("pliczek.txt")
# zawartosc = f.read()
# f.close()
# print(zawartosc)

# Funkcja open() nawiązuje połączenie z plikiem, ale nie wczytuje danych.
# Przypisuje jedynie do obiektu f wskaźnik do pliku. Wynika to z faktu że zbiór
# danych może być bardzo duży i programista powinien zachować kontrolę nad
# jego wczytywaniem. Funkcja read() wczytuje całą zawartość ze źródła f
# (pliku) do zmiennej zawartość. Następnie źródło zostaje zamknięte.

# Funkcja open()
# Podstawowa składnia tej funkcji jest następująca:

# file = open('sciezka_do_pliku', 'tryb', encoding=None)

# ▶ 'r' – tryb odczytu (domyślny), pozwala na czytanie zawartości pliku.
# ▶ 'r+' – tryb odczytu i modyfikacji
# ▶ 'w' – tryb zapisu, tworzy nowy plik lub nadpisuje istniejący.
# ▶ 'a' – tryb dopisywania, dodaje nowe dane na końcu pliku bez nadpisywania
# istniejącej zawartości.
# ▶ 'x' – otwarte do wyłącznego tworzenia, kończy się niepowodzeniem, jeśli
# plik już istnieje
# ▶ 'x+' – otwarte do wyłącznego tworzenia z możliwością odczytywania

# Każdy z tych trybów (oprócz ‚b') domyślnie otwiera plik do zapisu jako plik
# tekstowy. Jeżeli chcemy zapisywać plik formie binarnej musimy uzupełnić
# atrybut otwarcia o literę 'b', na przykład

# ▶ 'b' – tryb binarny, używany do pracy z plikami binarnymi.

# Zwykle pliki są otwierane w trybie tekstowym, co oznacza, ze odczytujesz i
# zapisujesz z i do pliku ciągi znaków, które są zakodowane w określonym
# kodowaniu. Jeśli kodowanie nie jest określone, domyślne jest zależne od
# platformy. Ponieważ UTF-8 jest współczesnym standardem de facto, zaleca się
# encoding="utf-8", chyba że wiesz, że musisz użyć innego kodowania. Dodanie
# „b" do trybu otwiera plik w trybie binarnym. Dane w trybie binarnym są
# odczytywane i zapisywane jako obiekty bajtów. Nie możesz określić kodowania
# podczas otwierania pliku w trybie binarnym.

# W trybie tekstowym, domyślnym ustawieniem podczas czytania jest konwersja
# zakończeń wierszy specyficznych dla platformy (\n) w systemie Unix, \r, \n w
# systemie Windows) na samo \n. Podczas pisania w trybie tekstowym,
```

```
# domyślnym ustawieniem jest konwersja wystąpień \n z powrotem na
# zakończenia wierszy specyficzne dla platformy. Ta modyfikacja danych pliku w
# tle jest w porządku dla plików tekstowych, ale uszkodzi dane binarne, takie jak
# te w plikach JPEG lub EXE. Należy zachować szczególną ostrożność podczas
# korzystania z trybu binarnego podczas czytania i zapisywania takich plików.


# Metoda close():

# Metoda close() zamyka otwarty plik.
# Zawsze powinieneś zamykać swoje pliki! W niektórych przypadkach, z powodu
# buforowania, zmiany wprowadzone do pliku mogą nie być widoczne, dopóki nie
# zamkniesz pliku.


# with

# Używanie with podczas pracy z obiektami plików należy do dobrych praktyk.
# Zaletą tego podejścia jest to, że plik jest prawidłowo zamykany po zakończeniu
# jego bloku, nawet jeśli w pewnym momencie zostanie zgłoszony wyjątek.
# Użycie with jest również znacznie krótsze niż pisanie równoważnych bloków try -
finally:

# with open("pliczek.txt", encoding="utf-8") as f:
#     read_data = f.read()
#     print(read_data)

# Możemy sprawdzić, że plik został automatycznie zamknięty.
# print(f.closed)

# Uwaga! Wywołanie f.write() bez użycia with lub f.close() może spowodować,
# że argumenty f.write() nie zostaną w pełni zapisane na dysku, nawet jeśli
# program zakończy się pomyślnie.

# Po zamknięciu obiektu pliku, zarówno przez instrukcję with, jak i f.close(),
# wszystkie próby użycia obiektu pliku automatycznie się nie powiodą.


# Metody obiektów plików:

# Aby odczytać zawartość pliku, należy wywołać polecenie f.read(size), które
# odczytuje pewną ilość danych i zwraca je jako ciąg znaków (w trybie
# tekstowym) lub obiekt bajtowy (w trybie binarnym). size jest opcjonalnym
# argumentem numerycznym. Gdy size jest pominięty lub ujemny, zostanie
# odczytana i zwrócona cała zawartość pliku. W przeciwnym razie odczytane i
# zwrócone zostanie co najwyżej size znaków (w trybie tekstowym) lub size
# bajtów (w trybie binarnym). Jeśli został osiągnięty koniec pliku, f.read() zwróci
# pusty ciąg znaków (”).


#wczytywanie linia po lini:

# W przypadku dużych plików lepiej zastosować procedurę czytania linia po linii.
# Czytanie linia po linii jest też operacją stosowaną, gdy chcemy odczytać z pliku
# konkretne linie:

# f = open("pliczek.txt")
#
# text = f.readline()
# print(text, end="*\n")
#
# text = f.readline()
# print(text, end="*\n")
#
# text = f.readline()
# print(text, end="*\n")
```

```
#
# f.close()
# print(f.closed)


# Wczytywanie po linii - to samo z with

# with open("pliczek.txt") as f:
#     text = f.readline()
#     print(text, end="*\n")
#     text = f.readline()
#     print(text, end="*\n")
#     text = f.readline()
#     print(text, end="*\n")
#
# f.close()
# print(f.closed)


# Wczytywanie linii do końca pliku w liście:

# f = open("pliczek.txt")
# text = f.readlines()
# print(text) # ['abrakadabra\n', 'abra\n', 'kadabra']
# f.close()

# Lub przy pomocy pętli:

# f = open("pliczek.txt")
# text = 1
# while text:
#     text = f.readline()
#     print(text, end="*")
# f.close()


# Wczytywanie linii do końca pliku w listą, to samo z with

# with open("pliczek.txt") as f:
#     text = f.readlines()
# print(text)

# Lub przy pomocy pętli:
# with open("pliczek.txt") as f:
#     text = 1
#     while text:
#         text = f.readline()
#         print(text, end="*")

# Jeszcze wczytywanie po linii w pętli:

# with open("pliczek.txt", "r") as file:
#     for line in file:
#         print(line.strip())

# Funkcja strip() usuwa znaki białe (takie jak spacje i nowe linie) z początku i
# końca każdej linii, co jest przydatne do czyszczenia danych. Wszystkie użyte
# funkcje są funkcjami systemowymi wbudowanymi w interpreter języka, zatem
# nie musimy importować żadnych dodatkowych bibliotek.

# Od teraz będę używał with open, zamiast close(), open()

# Zapisywanie danych do pliku:

# Zapis danych do pliku w Pythonie jest równie prosty jak ich odczyt. Aby
# zapisać dane do pliku, musimy otworzyć plik w trybie zapisu ('w') lub
```

```
# dopisywania (’a’). Warto nie mylić tych dwóch trybów – w pierwszym z nich
# jeżeli wskazany plik istnieje jego zawartość zostanie usunięta i zastąpiona przez
# nową. W drugim przypadku nowe dane zostaną dodane na końcu pliku. W obu
# przypadkach, jeżeli wskazany plik nie istnieje, zostanie on utworzony. Poniżej
# przykładu kodu w obu trybach:

# with open("output.txt", "w") as f:
#     f.write("To jest przykładowy tekst zapisany do pliku")
#
# with open("output.txt", "a") as f:
#     f.write("\nDodajemy nową linijkę tekstu")

# Zapisywanie danych do pliku ’x’:

# Jest jeszcze jeden warty uwagi a mało znany tryb zwany Exclusive creation. Za
# jego pomocą otworzymy plik do zapisu, ale tylko jeśli plik nie istnieje. Jeśli plik
# istnieje, Python zgłosi błąd FileExistsError.

# with open('output1.txt', 'x') as file:
#     file.write('To jest przykladowy tekst zapisany do pliku.')
#
# with open("pliczek.txt", "x") as file:
#     file.write('To jest przykladowy tekst zapisany do pliku.') #PRAWDA TO!!!


#writelines:
# Metoda writelines() zapisuje elementy listy do pliku.
# Miejsce, w którym zostaną wstawione teksty, zależy od trybu pliku i pozycji
# strumienia.

# ▶ ’a’ – Teksty zostaną wstawione w bieżącej pozycji strumienia pliku,
# domyślnie na końcu pliku.

# ▶ ’w’ – Plik zostanie opróżniony przed wstawieniem tekstów w bieżącej
# pozycji strumienia pliku, domyślnie 0.

# with open("pliczek.txt", "a") as file:
#     file.writelines(["See you soon!", "Over and out."])
#
# #open and read the file after the appending:
# with open("pliczek.txt", "r") as f:
#     print(f.read())


# Metoda seek():

# Metoda seek() ustawia bieżącą pozycję pliku w strumieniu plików

# with open("pliczek.txt", "r") as file:
#     a = file.seek(5)
#     print(a)
#     print(file.readline())

# Metoda seekable() zwraca True, jeśli plik jest wyszukiwalny, False, jeśli nie.
# Plik jest wyszukiwalny, jeśli umożliwia dostęp do strumienia plików, tzn metoda
# seek().


# Odczyt i zapis

# Istnieją też tryby mieszane, które łączą tryby tekstowe z trybami binarnymi oraz
# operacje zapisu i odczytu. Dzięki nim możemy wykonywać bardziej
# zaawansowane operacje na plikach, które wymagają zarówno czytania, jak i
# modyfikowania zawartości pliku bez konieczności zamykania i ponownego
# otwierania go w innym trybie. Tryby mieszane są szczególnie przydatne w
```

```
# scenariuszach, gdzie dane muszą być dynamicznie aktualizowane lub
# weryfikowane, a następnie zapisywane z powrotem do tego samego pliku. Tryby
# mieszane poznamy po tym, że w atrybutach ich otwarcia znajduje się znak ‘+’.
# Przykładowo:

# with open("pliczek.txt", "r+") as file:
#     content = file.read()
#     file.write("Dopisana wartosc")

# Powyższe kod otwiera plik do odczytu i zapisu. Jeśli plik nie istnieje, Python
# zgłosi błąd FileNotFoundError.

# Po odczytywaniu wskaźnik jest na końcu pliku, i treść do zapisywania zapisuje
# się na koniec.


# Gdzie się zapisujemy?

# Porównaj z poprzednim:

# ▶ Treść do zapisywania zapisuje się na początku pliku.

# with open("pliczek.txt", "r+") as file:
#     file.write("Dopisana zawartosc")
#
# with open("pliczek.txt", "r+") as file:
#     file.seek(5)
#     file.write("Dopisana zawartosc")

# Metoda seek() ustawia bieżącą pozycję pliku w strumieniu plików

# with open("plik.txt", "r+") as f:
#     a = f.seek(5)
#     f.write("See you soon!")
#     f.seek(0)
#     print(f.read())

# with open("plik.txt", "r+") as f:
#     a = f.seek(5)
#     print(a)
#     print(f.write('See you soon!'))
#     print(f.read())
#     f.seek(0)
#     print(f.read())


# Inne metody:

# ▶ Metoda tell() zwraca bieżącą pozycję pliku w strumieniu plików.
# ▶ Metoda readable() zwraca True, jeśli plik jest czytelny, lub False w
# przeciwnym razie.
# ▶ Metoda writable() zwraca True, jeśli do pliku można zapisywać, lub False
# w przeciwnym razie.


#truncate():
# ▶ Metoda truncate() zmienia rozmiar pliku do podanej liczby bajtów.

# with open("plik.txt", "a") as file:
#     file.truncate(10)
#
# #open and read the file after the truncate:
# with open("plik.txt", "r") as file:
#     print(file.read())
```

```
# Przykład:
# Źródło:
# https://www.flynerd.pl/2018/01/python-metody-typu-string.html

# Wyobraź sobie, że jesteś bioinformatykiem i otrzymujesz kod genetyczny do
# analizy w pliku tekstowym.

# Kod DNA składa się z 4 zasad azotowych: adeniny(A), cytozyny(D),
# guaniny(G), tyminy(T). Idealny kod DNA wygląda następująco:
# TGCACGATCATGTCTACTATCCTCTCTATGGTGGGGTT. . .

# Zdarza się, jednak, że kod zawiera małe jak i duże litery. Kolejny problem to
# maszyny sekwencjonujące nie są wolne od błędów. W zależności od maszyny
# błędy sekwencjonowania mogą zostać zamienione na znak – czy literę N.

# ▶ wczytaj plik
# ▶ policz ile razy występuje w kodzie każda zasada azotowa - adenina, cytozyna, guanina,
tymina.


# with open("ohno.txt", "r") as file:
#     seq0 = file.read()
#
# seq = seq0.strip().upper()
# print(seq)
# n = len(seq)
# occureness = {}
# for x in "ACTG":
#     occureness[x] = seq.count(x)
#
# print(occureness)

#Dzięki Murka

# Przykład dalej:

# W dokumentacji znajduje się następujący zapis:
# gdy jakość sekwencji nie pozwala dokładnie odczytać rodzaju zasady azotowej
# wstawiany jest znak „-” Natomiast, gdy laser sczytujący ześlizgnie się,
# wstawiane są litery „N”, jednocześnie ostatnia wartość zasady jest ponownie
# odczytywana bez ubytku zasady w tym miejscu.
# Co za przydatna informacja!

# ▶ Oczyść DNA z błędów typu N.
# ▶ Policz wystąpienia sekwencji GAGA
# ▶ Znajdź miejsce (indeks) w łańcuchu, gdzie występuje 7 guanin z rzędu
# ▶ Znajdź miejsce (indeks) , gdzie od końca łańcucha występuje 6 cytozyn
# ▶ Policz ile razy w kodzie pojawiła się sekwencja CTGAAA
# ▶ W sekwencji CTGAAA czasem mutuje ostania litera A, wówczas jakość
# ostatniej litery może być wątplia. Ile sekwencji znajdziesz, jeśli weźmiesz
# pod uwagę wątpliwą, ostatnią adeninę?
# ▶ Na podstawie czystej nici utwórz odpowiadającą jej nić RNA (nić RNA w
# miejscu tyminy będzie mieć uracyl (U)). Nic RNA zapisz do nowego pliku
# RNA.txt

# with open("ohno.txt", "r") as file:
#     seq0 = file.read()
#
# seq = seq0.strip().upper()
# print(seq)
# n = len(seq)
# DNA = seq.replace("N", '')
# GAGA = DNA.count("GAGA")
# print("Liczba wystąpień sekwencji GAGA:", GAGA)
```

```
#
# CTGAAA = DNA.count("CTGAAA")
# print("Liczba wystąpień sekwencji CTGAAA:", CTGAAA)
#
# CTGAA_ = DNA.count("CTGAA_")
# print("Liczba wystąpień sekwencji CTGAAA i CTGAA_", CTGAAA + CTGAA_)
#
# RNA = DNA.replace("T", "U")
# with open("RNA.txt", "w") as f:
#     f.write(RNA)



#Klasy:

# Pusta klasa:
# class Cookie:
#     pass
#
# cookie = Cookie()

# Zmienne obiektu:

# Do obiektu możemy przypisać zmienną z określoną wartością. Taka wartość
# dotyczyć będzie wyłącznie tego jednego obiektu. Aby odnieść się do zmiennej
# w obiekcie, musimy wskazać zarówno obiekt, jak i zmienną, a oddzielamy ich
# nazwy kropką. Dla przykładu, aby odnieść się do zmiennej type w obiekcie
# cookie1, użyjemy cookie1.type. Zarówno do przypisania wartości, jak i do jej
# pobrania.

# class Cookie:
#     pass
#
# cookie1 = Cookie()
# cookie2 = Cookie()
#
# cookie1.type = "Biscuit"
# cookie1.color = "White"
# cookie2.type = "Oreo"
#
# print(cookie1.type)
# print(cookie1.color)
# print(cookie2.type)

# Nieczęsto tworzy się zmienne obiektu tak jak w powyższym przykładzie: poza
# klasą. Często jest to wręcz uznawane za złą praktykę. Częściej tworzy się je w
# obrębie metod, a zwłaszcza szczególnej metody zwanej inicjalizatorem.
# Gdy tworzymy nowy obiekt, stawiamy nawias za nazwą klasy. Ten nawias to
# wywołanie funkcji tworzącej obiekt, zwanej konstruktorem. Funkcja ta
# przechodzi przez szereg kroków, niezbędnych do utworzenia obiektu, w tym
# między innymi woła specjalną metodę o nazwie __init__ z naszej klasy. Ta
# metoda zwana jest inicjalizatorem. W jej ciele określamy, co powinno się dziać
# w czasie tworzenia obiektu. Najczęściej definiujemy w niej atrybuty obiektu.

# class Cookie:
#     def __init__(self, type, color):
#         self.type = type
#         self.color = color
#
# cookie1 = Cookie("Biscuit", "White")
# print(cookie1.type)
# print(cookie1.color)

# class Cookie:
#     def __init__(self, type, color = None):
```

```
#         self.type = type
#         self.color = color
#
# cookie1 = Cookie("Biscuit","White")
# cookie2 = Cookie("Oreo")
# print(cookie1.type) # Biscuit
# print(cookie1.color) # White
# print(cookie2.type) # Oreo
# print(cookie2.color) #None


#Metody:

# Wewnątrz klas możemy definiować funkcje. Takie funkcje nazywane są
# metodami. Definiujemy je w ciele klasy, a ich pierwszym parametrem jest
# odniesienie do instancji obiektu, na którym tę metodę wywołamy. Parametr ten
# powinno nazywać się self. Gdy wywołujemy metodę, zaczynamy od obiektu,
# następnie stawiamy kropkę, nazwę metody i nawias z argumentami.

# class Position:
#     def __init__(self, x = 0.0, y = 0.0):
#         self.x = x
#         self.y = y
#     def step_right(self):
#         self.x += 1.0
#     def move_up(self, value):
#         self.y += value
#
# pos = Position()
# pos.step_right()
# print(pos.x) # 1.0
# pos.move_up(6.0)
# print(pos.y) # 6.0
# pos.move_up(3.0)
# print(pos.y) # 9.0

#Obiekty i zmienne:

# Każdy obiekt jest osobnym bytem. To, że wyglądają podobnie, nie znaczy, że
# mają na siebie wpływ. Dlatego też w poniższym przykładzie zmiana name w
# obiekcie user1 nie będzie miała żadnego wpływu na user2.

# class User:
#     def __init__(self, name):
#         self.name = name

# user1 = User("Rafał")
# user2 = User("Rafał")
# print(user1.name) # Rafał
# print(user2.name) # Rafał
# user1.name = "Bartek"
# print(user1.name) # Bartek
# print(user2.name) # Rafał


# Z drugiej strony, jeśli mamy dwie zmienne wskazujące na jeden obiekt, to
# możemy go zmienić przy użyciu dowolnej z nich. Po takim zabiegu, wartości
# dla obu zmiennych ulegną zmianie, bo przecież przekształcone zostało coś, na
# co obydwie wskazują.

# user1 = User("Rafał")
# user2 = user1
# print(user1.name)
# # Rafał
# print(user2.name)
# # Rafał
```

```
# user1.name = "Bartek"
# print(user1.name)
# # Bartek
# print(user2.name)
# # Bartek


# Dwie zmienne wskazują na ten sam obiekt i właściwość tego obiektu zmienia
# wartość.
# Warto porównać to z przykładem, gdy dwa obiekty pokazywały na tą samą
# wartość, a potem zmieniło się to, na co jedna z tych zmiennych wskazuje.
# Wynik będzie inny.

# user1 = User("Rafał")
# user2 = user1
# print(user1.name)
# # Rafał
# print(user2.name)
# # Rafał
# user1 = User("Bartek")
# print(user1.name)
# # Bartek
# print(user2.name)
# # Rafał


# Dwie zmienne wskazują na ten sam obiekt i właściwość tego obiektu zmienia
# wartość.

# class User:
#     def __init__(self, name):
#         self.name = name
#     def copy(self):
#         return User(self.name)
#
# user1 = User("Rafał")
# user2 = user1.copy()
# user1.name = "Bartek"
# print(user1.name) # Bartek
# print(user2.name) # Rafał
# user2.name = "Marek"
# print(user1.name) # Bartek
# print(user2.name) # Marek


#Metoda __str__:
# class User:
#     def __init__(self, name):
#         self.name = name
#     def __str__(self):
#         return f'Username: {self.name}'
# user1 = User("Rafał")
# print(user1)


# Dziedziczenie(inheritance)

# class Person:
#     def __init__(self, fname, lname):
#         self.firstname = fname
#         self.lastname = lname
#     def printname(self):
#         print(self.firstname, self.lastname)
#
# class Student(Person):
#     pass
```

```
#
# x = Person("John", "Doe")
# x.printname()
# x = Student("Mike", "Olsen")
# x.printname()


# class Person:
#     def __init__(self, fname, lname):
#         self.firstname = fname
#         self.lastname = lname
#
#     def printname(self):
#         print(self.firstname, self.lastname)
#
# class Student(Person):
#     def __init__(self, fname, lname, number):
#         Person.__init__(self, fname, lname)
#         # super().__init__(fname, lname)
#         self.number = number
#
# x = Person("John", "Doe")
# x.printname()
# x = Student("Mike", "Olsen", 1999)
# x.printname()

# class Parent:
#   def __init__(self, txt):
#     self.message = txt
#
#   def printmessage(self):
#     print(self.message)
#
# class Child(Parent):
#   def __init__(self, txt):
#     super().__init__(txt)
#
# x = Child("Hello, and welcome!")
#
# x.printmessage()




# G!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```