Dokument ten jest jawną pulą zadań z przedmiotu **Wprowadzenie do programowania w języku Python**. Zadania zostały podzielone na cztery kategorie:

- Funkcje - podstawy,
- Instrukcje warunkowe,
- Pętle,
- Listy,
- Słowniki.

Każde zadanie opiera się na zdefiniowaniu funkcji działającej w określony sposób. Do każdego zadania dodane są przykłady jej działania - upewniając się, że zdefiniowana przez Ciebie funkcja zwraca odpowiednie wyniki dla tych przykładów, możesz być prawie pewn[ay], że zadanie zostało rozwiązane prawidłowo. Założenie jest następujące - rozwiązując zadania z danej sekcji zakładamy, że wcześniejszy materiał został już opanowany, dlatego w pętlach przy niektórych zadaniach konieczna jest znajomość instrukcji warunkowych, a w listach i słownikach pętli oraz instrukcji warunkowych. Powodzenia!

---

1. Funkcje - podstawy

1.1. cubic_sum

Zdefiniuj funkcję o nazwie **cubic_sum**, która przyjmuje trzy argumenty będące liczbami rzeczywistymi i zwraca sumę ich sześcianów.
Przykłady:

```
cubic_sum(1, 2, 3)
36

cubic_sum(5, 9, -2)
846

cubic_sum(-3.5, -12.29, 7.2)
-1525.9589889999995
```

1.2. power_of_sum

Zdefiniuj funkcję o nazwie **power_of_sum**, która przyjmuje trzy argumenty, pierwsze dwa to dowolne liczby rzeczywiste, natomiast trzeci to liczba całkowita, będąca potęgą. Funkcja powinna zwrócić potęgę określonego stopnia sumy pierwszych dwóch liczb, tzn. wyznaczać wynik działania:

$$(a+b)^n.$$

Przykłady:

```
power_of_sum(2, 3, 5)
3125

power_of_sum(-2, 2.4, -10)
9536.743164062522

power_of_sum(5, -15, 7)
-10000000
```

1.3. delta

Zdefiniuj funkcję o nazwie delta, która przyjmuje trzy argumenty będące liczbami rzeczywistymi - współczynnikami a, b, c funkcji kwadratowej o wzorze:

$$ax^2 + bx + c.$$

Funkcja powinna zwracać wartość delty funkcji kwadratowej, tj. wynik działania:

$$b^2 - 4ac.$$

Przykłady:

```
delta(1, 2, 1)
0

delta(-0.1, 231, -1500)
52761.0

delta(-1, -2, -2)
-4
```

1.4. profit

Zdefiniuj funkcję o nazwie **profit**, która przyjmuje jeden argument będący liczbą naturalną dodatnią. Pewna firma wyznaczyła wzór na zyski ze sprzedaży produkowanego dobra, dany następującą formułą:

$$-0.1x^2 + 231x - 1500,$$

gdzie x jest ilością wyprodukowanego dobra. Funkcja powinna zwracać wielkość przychodu w zależności od tej ilości, czyli wartość tej kwadratowej funkcji. Przykłady:

```
>>> profit(0)
-1500.0

profit(100)
20600.0

profit(1000)
129500.0

profit(10000)
-7691500.0
```

1.5. product_and_price

Zdefiniuj funkcję o nazwie **product_and_price**, która przyjmuje dwa argumenty - nazwę kupowanego produktu oraz jego cenę. Funkcja powinna tekst konstruowany następująco:

```
{product} was purchased for {price}
```

Przykłady:

```
>>> product_and_price('banana', '5.29')
'banana was purchased for 5.29'
>>> product_and_price('chopped tomatoes', '3.59')
'chopped tomatoes was purchased for 3.59'
```

1.6. euc_distance

Zdefiniuj funkcję o nazwie **euc_distance**, która przyjmuje trzy argumenty będące liczbami rzeczywistymi `x, y, z` - współrzędnymi punktu w trójwymiarowym układzie współrzędnych. Funkcja powinna zwracać odległość takiego punktu od początku układu współrzędnych. Odległość ta dana jest wzorem:

$$\sqrt{x^2 + y^2 + z^2}.$$

Przykłady:

```
euc_distance(0, 0, 0)
0

euc_distance(1, 1, 1)
1.7320508075688772

euc_distance(-4.2, 0.3, 5.9)
7.2484481097680495

euc_distance(5, 12, -20)
23.853720883753127
```

1.7. cuboid

Zdefiniuj funkcję o nazwie **cuboid**, która przyjmuje trzy argumenty będące dodatnimi liczbami rzeczywistymi - kolejno długości boków podstawy prostopadłościanu i jego wysokości. Funkcja powinna zwracać pole powierzchni oraz objętość takiego prostopadłościanu. Przykłady:

```
cuboid(1, 1, 1)
(6, 1)

cuboid(5, 3, 8)
(158, 120)

cuboid(2.34, 6.02, 4.89)
(109.93439999999998, 68.88445199999998)
```

Wzory na pole powierzchni i objętość są następujące:

$$P = 2(ab + ah + bh),$$
$$V = abh.$$

1.8. insert_at

Odwoływanie się do konkretnego znaku w stringu po indeksie działa tak samo jak w przypadku list. Mając zapisane pod zmienną `breed` string `labrador`, możemy odwoływać się do poszczególnych elementów i wycinków następująco:

```
breed[0]
'l'

breed[-1]
'r'

breed[4]
'a'

breed[1:]
'abrador'

breed[2:5]
'bra'
```

Zdefiniuj funkcję o nazwie **insert_at**, która przyjmuje trzy argumenty - dwa stringi i liczbę naturalną i. Funkcja powinna zwrócić zmodyfikowany pierwszy ze stringów w taki sposób, że drugi dodany jest do pierwszego na miejscu o indeksie i. Przykłady:

```
insert_at('Pyhon', 't', 2)
'Python'

insert_at('ord', 'w', 0)
'word'

insert_at('Lazar', 'ski', 5)
'Lazarski'

insert_at('Philodron', 'den', 5)
'Philodendron'
```

2. Instrukcje warunkowe

2.1. even_odd

Zdefiniuj funkcję o nazwie **even_odd**, która przyjmuje jeden argument będący liczbą całkowitą. Funkcja zwraca `even`, jeśli liczba jest parzysta, `odd` w przeciwnym przypadku. Przykłady:

```
even_odd(2)
'even'

even_odd(0)
'even'

even_odd(-30)
'even'

even_odd(1)
'odd'

even_odd(123)
'odd'
```

2.2. delta_int

Zdefiniuj funkcję o nazwie **delta_int**, która przyjmuje trzy argumenty będące liczbami rzeczywistymi. Funkcja po wyznaczeniu delty powinna drukować informację czy delta jest dodatnia, ujemna czy równa zero w formie:

```
Delta is zero
Delta is positive
Delta is negative
```

Przykłady:

```
delta_int(1, 2, 1)
'Delta is zero'

delta_int(-0.1, 231, -1500)
'Delta is positive'

delta_int(-1, -2, -2)
'Delta is negative'
```

2.3. in_range

Zdefiniuj funkcję o nazwie **in_range**, która przyjmuje trzy argumenty będące liczbami rzeczywistymi. Funkcja powinna weryfikować, czy pierwszy argument jest liczbą leżącą w przedziale skonstruowanym przez dwie ostatnie liczby. Funkcja zwraca `True` jeśli liczba leży wewnątrz przedziału, `False` w przeciwnym przypadku. Przykłady:

```
in_range(1, 0, 2)
True

in_range(-2.5, -5, 5)
True

in_range(4, 0, 4)
False

in_range(-1, -1, -2)
False
```

2.4. divisibility

Zdefiniuj funkcję o nazwie **divisibility**, która przyjmuje dwa argumenty będące liczbami całkowitymi. Funkcja powinna zwracać `True`, jeśli pierwszy argument jest podzielny przez drugi, `False` w przeciwnym przypadku. Przykłady:

```
divisibility(0, 2)
```

```
True

divisibility(15, 3)
True

divisibility(-49, 7)
True

divisibility(1, 100)
False

divisibility(-33, 2)
False
```

2.5. max_from_3

Zdefiniuj funkcję o nazwie **max_from_3**, która przyjmuje trzy liczby rzeczywiste i zwraca największą z nich. Przykłady:

```
max_from_3(1, 2, 3)
3

max_from_3(-5, -2, -10)
-2

max_from_3(39, -12, 20)
39
```

2.6. calculate

Zdefiniuj funkcję o nazwie **calculate**, która przyjmuje trzy argumenty - pierwsze dwa to liczby rzeczywiste, a trzeci jest jednym z następujących stringów: `sum`, `multi`, `diff`, `div` oraz `pow`. Funkcja powinna zwracać wynik przeprowadzenia określonej operacji na pierwszych dwóch argumentach. Przykłady:

```
calculate(5, -7, 'sum')
```

```
-2

calculate(3, 2, 'multi')
6

calculate(48, -20, 'diff')
68

calculate(-371, 42, 'div')
-8.833333333333334

calculate(5, -3, 'pow')
0.008
```

2.7. prefix_slice

Zdefiniuj funkcję o nazwie **prefix_slice**, która przyjmuje dwa argumenty będące stringami. Funkcja powinna zwracać `True`, jeśli drugi argument jest prefixem pierwszego, `False` w przeciwnym wypadku. Funkcja musi wykorzystywać w tym celu odwoływanie się po indeksie do wycinka łańcucha znaków. Przykłady:

```
prefix_slice('word', 'wo')
True

prefix_slice('word', 'or')
False

prefix_slice('labrador', 'dog')
False

prefix_slice('autobiography', 'auto')
True

prefix_slice('circumnavigate', 'anti')
False
```

2.8. leap_check

Zdefiniuj funkcję o nazwie **leap_check**, która przyjmuje jeden argument będący liczbą całkowitą, którą utożsamiamy z rokiem. Funkcja zwraca `True` jeśli rok jest przystępny, `False` w przeciwnym przypadku. Rok przystępny zazwyczaj wypada co cztery lata, jednak są wyjątki od tej reguły. W przypadku pełnych stuleci jak 1900, 2000, 2100 rok jest przystępny tylko wtedy, gdy jest podzielny przez 400, nie wystarczy wyłącznie zasada podzielności przez 4. Pseudokod struktury funkcji można rozpisać następująco:

```
if year is divisible by 4 and not by 100 or year is
divisible by 400 -> year is a leap year
if not -> year is not a leap year
```

Przykłady:

```
leap_check(1900)
False

leap_check(2000)
True

leap_check(2024)
True

leap_check(1970)
False

leap_check(2023)
False
```

---

3. Pętle

3.1. factorial

Zdefiniuj funkcję o nazwie **factorial**, która przyjmuje jeden argument będący liczbą naturalną dodatnią i zwraca silnię tej liczby. Przykłady:

```
factorial(1)
1

factorial(5)
120

factorial(10)
3628800

factorial(25)
15511210043330985984000000
```

3.2. divisors

Zdefiniuj funkcję o nazwie **divisors**, która przyjmuje jeden argument będący liczbą naturalną dodatnią i wyświetla kolejno wszystkie dzielniki tej liczby. Na przykład:

```
divisors(1)
1

divisors(2)
1
2

divisors(8)
1
2
4
8

divisors(120)
1
2
3
4
```

```
5
6
8
10
12
15
20
24
30
40
60
120
```

3.3. prefix_for

Zdefiniuj funkcję o nazwie **prefix_for**, która przyjmuje dwa argumenty będące stringami. Funkcja powinna zwracać `True`, jeśli drugi argument jest prefixem pierwszego, `False` w przeciwnym wypadku. Funkcja musi wykorzystywać w tym celu pętle. Przykłady:

```
prefix_for('word', 'wo')
True

prefix_for('word', 'or')
False

prefix_for('labrador', 'dog')
False

prefix_for('autobiography', 'auto')
True

prefix_for('circumnavigate', 'anti')
False
```

3.4. str_reverse

Zdefiniuj funkcję o nazwie **str_reverse**, która przyjmuje jeden argument będący stringiem. Funkcja powinna zwracać string zapisany od tyłu. Przykłady:

```
str_reverse('dog')
'god'

str_reverse('cat')
'tac'

str_reverse('dinosaur')
'ruasonid'

str_reverse('Was it a car or a cat I saw?')
'?was I tac a ro rac a ti saW'

str_reverse('kayak')
'kayak'
```

3.5. divisible_from_range

Zdefiniuj funkcję o nazwie **divisible_from_range**, która przyjmuje trzy argumenty będące liczbami całkowitymi. Pierwszy z nich jest dzielnikiem n, z kolei kolejne określają domknięty przedział $[a, b]$. Funkcja powinna drukować wszystkie liczby podzielne przez n, które leżą w przedziale $[a, b]$. Przykłady:

```
divisible_from_range(3, 11, 25)
12
15
18
21
24

divisible_from_range(5, 15, 30)
15
20
25
```

```
30

divisible_from_range(4, 9, 35)
12
16
20
24
28
32

divisible_from_range(1, 0, 5)
0
1
2
3
4
5
```

3.6. rectangle

Zdefiniuj funkcję o nazwie **rectangle**, która przyjmuje dwa argumenty będące dodatnimi liczbami naturalnymi r, c. Funkcja powinna drukować prostokąt o wymiarach $r \times c$ złożony z `*`. Przykłady:

```
rectangle(1, 1)
*

rectangle(1, 4)
****

rectangle(3, 4)
****
****
****

rectangle(5, 15)
***************
***************
***************
```

```
***************
***************
```

3.7. triangle

Zdefiniuj funkcję o nazwie **triangle**, która przyjmuje jeden argument będący dodatnią liczbą naturalną h. Funkcja powinna drukować trójkąt o wysokości h, który powinien być równoramienny, zaczynać się od wierzchołka z jedną gwiazdką `*` i wraz z kolejnym wierszem powiększać swoją długość o dwie gwiazdki. Przykłady:

```
triangle(0)



triangle(1)
 *

triangle(3)
    *
   ***
  *****

triangle(10)
           *
          ***
         *****
        *******
       *********
      ***********
     *************
    ***************
   *****************
  *******************
```

3.8. del_all

Zdefiniuj funkcję o nazwie **del_all**, która przyjmuje dwa argumenty będące stringami, z czego drugi powinien mieć długość jeden. Funkcja powinna zwrócić pierwszego stringa z usuniętymi wszystkimi wystąpieniami argumentu drugiego. Przykłady:

```
del_all('word', 'w')
'ord'

del_all('word', 'x')
'word'

del_all('pseudopseudohypoparathyroidism', 'e')
'psudopsudohypoparathyroidism'

del_all('bobby', 'b')
'oy'
```

---

4. Listy

4.1. product

Zdefiniuj funkcję o nazwie **product**, która przyjmuje jeden argument będący listą z liczbami rzeczywistymi i zwraca ich iloczyn. Przykłady:

```
product([1, 2, 3, 4])
24

product([-5, 2.35, -11, 6.1])
788.425

product([-100, 123, 91, 0])
0

product([5, -0.2])
-1.0
```

4.2. product_of_reverses

Zdefiniuj funkcję o nazwie **product**, która przyjmuje jeden argument będący listą z liczbami rzeczywistymi i zwraca ich iloczyn ich odwrotności. Przykłady:

```
product_of_reverses([1, 2, 3, 4])
0.041666666666666664

product_of_reverses([-5, 2.35, -11, 6.1])
0.0012683514601896189

product_of_reverses([5, -0.2])
-1.0

product_of_reverses([0.1, -0.0002, 0.5])
-100000.0
```

4.3. duplicates

Zdefiniuj funkcję o nazwie **duplicates**, która przyjmuje jeden argument będący listą i zwraca te elementy, które wystąpiły w liście więcej niż jeden raz. Przykłady:

```
duplicates([1, 1, 2, 3, 4, 4, 4, 5])
[1, 4]

duplicates(['cat', 'dog', 'dragon', 'dog', 'dog',
'dragon'])
['dog', 'dragon']
```

4.4. extend_list

Zdefiniuj funkcję o nazwie **extend_list**, która przyjmuje dwa argumenty - jeden będący listą i drugi będący liczbą naturalną dodatnią n. Funkcja powinna zwracać listę składającą się z n wejściowych list. Przykłady:

```
extend_list([1, 2, 3], 4)
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

extend_list(['cat', 'dog'], 3)
['cat', 'dog', 'cat', 'dog', 'cat', 'dog']
```

4.5. odd_numbers

Zdefiniuj funkcję o nazwie **odd_numbers**, która przyjmuje jeden argument będący listą z liczbami całkowitymi i zwraca wszystkie liczby rzeczywiste zawarte w tej liście. Przykłady:

```
odd_numbers([1,2,3,4,5,6])
[1, 3, 5]

odd_numbers([-5, 0, -2, 4, 1, -1, 9])
[-5, 1, -1, 9]
```

4.6. list_reverse

Zdefiniuj funkcję o nazwie **list_reverse**, która przyjmuje jeden argument będący listą. Funkcja powinna zwracać listę zapisaną od tyłu. Przykłady:

```
list_reverse([1, 2, 3, 4])
[4, 3, 2, 1]

list_reverse(['cat', 'dog', 'dragon'])
['dragon', 'dog', 'cat']
```

4.7. concatenate

Zdefiniuj funkcję o nazwie **concatenate**, która przyjmuje jeden argument będący listą ze stringami. Funkcja powinna połączyć ze sobą wszystkie stringi oddzielając je spacją. Przykłady:

```
concatenate(['dog', 'cat', 'dragon'])
'dog cat dragon'

concatenate(['The', 'quick', 'brown', 'fox', 'jumps',
'over', 'the', 'lazy', 'dog'])
'The quick brown fox jumps over the lazy dog'

concatenate(['5', '6'])
'56'
```

4.8. shortest_string

Zdefiniuj funkcję o nazwie **shortest_string**, która przyjmuje jeden argument będący listą ze stringami. Funkcja powinna zwrócić pierwszy najkrótszy string z listy. Przykłady:

```
shortest_string(['dragon', 'dog', 'cat', 'giraffe'])
'dog'

shortest_string(['Mark', 'Alice', 'Elliot', 'Robin'])
'Mark'
```

---

5. Słowniki

5.1. str_lengths

Zdefiniuj funkcję o nazwie **str_lengths**, która przyjmuje jeden argument będący listą ze stringami i zwraca słownik, w którym klucze są tymi stringami, a wartości ich długościami. Przykłady:

```
str_lengths(['Mark', 'Alice', 'Walter'])
{'Mark': 4, 'Alice': 5, 'Walter': 6}

str_lengths(['cat', 'dog', 'giraffe', 'bird'])
{'cat': 3, 'dog': 3, 'giraffe': 7, 'bird': 4}
```

5.2. even_parity

Zdefiniuj funkcję o nazwie **even_parity**, która przyjmuje jeden argument będący listą z liczbami całkowitymi. Funkcja powinna zwracać słownik, w którym kluczami są te liczby, a wartościami `even`, jeśli liczba jest parzysta i `odd` w przeciwnym przypadku. Przykłady:

```
even_parity([1, 2, 3, 4])
{1: 'odd', 2: 'even', 3: 'odd', 4: 'even'}

even_parity([-40, 31, -21, -102, 0, 62])
{-40: 'even', 31: 'odd', -21: 'odd', -102: 'even', 0:
'even', 62: 'even'}
```

5.3. sum_of_values

Zdefiniuj funkcję o nazwie **sum_of_values**, która przyjmuje jeden argument będący słownikiem i zwraca sumę wartości tego słownika. Przykłady:

```
sum_of_values({'Anna': 4, 'Alice': 5})
9

sum_of_values({'Poland': 37.75, 'Ukraine': 43.79,
'Czechia': 10.51})
92.05
```

5.4. min_from_dict

Zdefiniuj funkcję o nazwie **min_from_dict**, która przyjmuje jeden argument będący słownikiem i zwraca klucz, dla którego wartość jest najmniejsza. Przykłady:

```
min_from_dict({'Anna': 4, 'Alice': 5})
'Anna'

min_from_dict({'Poland': 37.75, 'Ukraine': 43.79,
'Czechia': 10.51})
'Czechia'
```

5.5. avg_from_dict

Zdefiniuj funkcję o nazwie **avg_from_dict**, która przyjmuje jeden argument będący słownikiem i zwraca średnią jego wartości.
Przykłady:

```
avg_from_dict({'Anna': 4, 'Alice': 5})
4.5

avg_from_dict({'Poland': 37.75, 'Ukraine': 43.79,
'Czechia': 10.51})
30.683333333333334
```

5.6. dict_val_to_list

Zdefiniuj funkcję o nazwie **dict_val_to_list**, która przyjmuje jeden argument będący słownikiem i zwraca jego wartości w formie listy.
Przykłady:

```
dict_val_to_list({'Anna': 4, 'Alice': 5})
[4, 5]

dict_val_to_list({'Poland': 37.75, 'Ukraine': 43.79,
'Czechia': 10.51})
[37.75, 43.79, 10.51]
```

5.7. create_dict

Zdefiniuj funkcję o nazwie **create_dict**, która przyjmuje dwa argumenty, oba będące listami. Zakładamy, że pierwsza lista zawiera klucze, a druga wartości i z tak przygotowanych danych funkcja powinna zwracać słownik. Przykłady:

```
create_dict(['Anna', 'Alice'], [4, 5])
{'Anna': 4, 'Alice': 5}

create_dict(['Poland', 'Ukraine', 'Czechia'], [37.75,
43.79, 10.51])
{'Poland': 37.75, 'Ukraine': 43.79, 'Czechia': 10.51}
```

5.8. sum_dict

Zdefiniuj funkcję o nazwie **sum_dict**, która przyjmuje dwa argumenty, oba będące słownikami z takimi samymi kluczami. Funkcja powinna zwracać słownik z tymi samymi kluczami i zsumowanymi wartościami z obu słowników. Przykłady:

```
sum_dict({'a': 4, 'b': -2}, {'a': -9, 'b': 12})
{'a': -5, 'b': 12}

sum_dict({'e': 123, 'f': -123, 'g': 2000}, {'f': 123, 'g':
-1000, 'e': 77})
{'e': 200, 'f': 0, 'g': 1000}
```