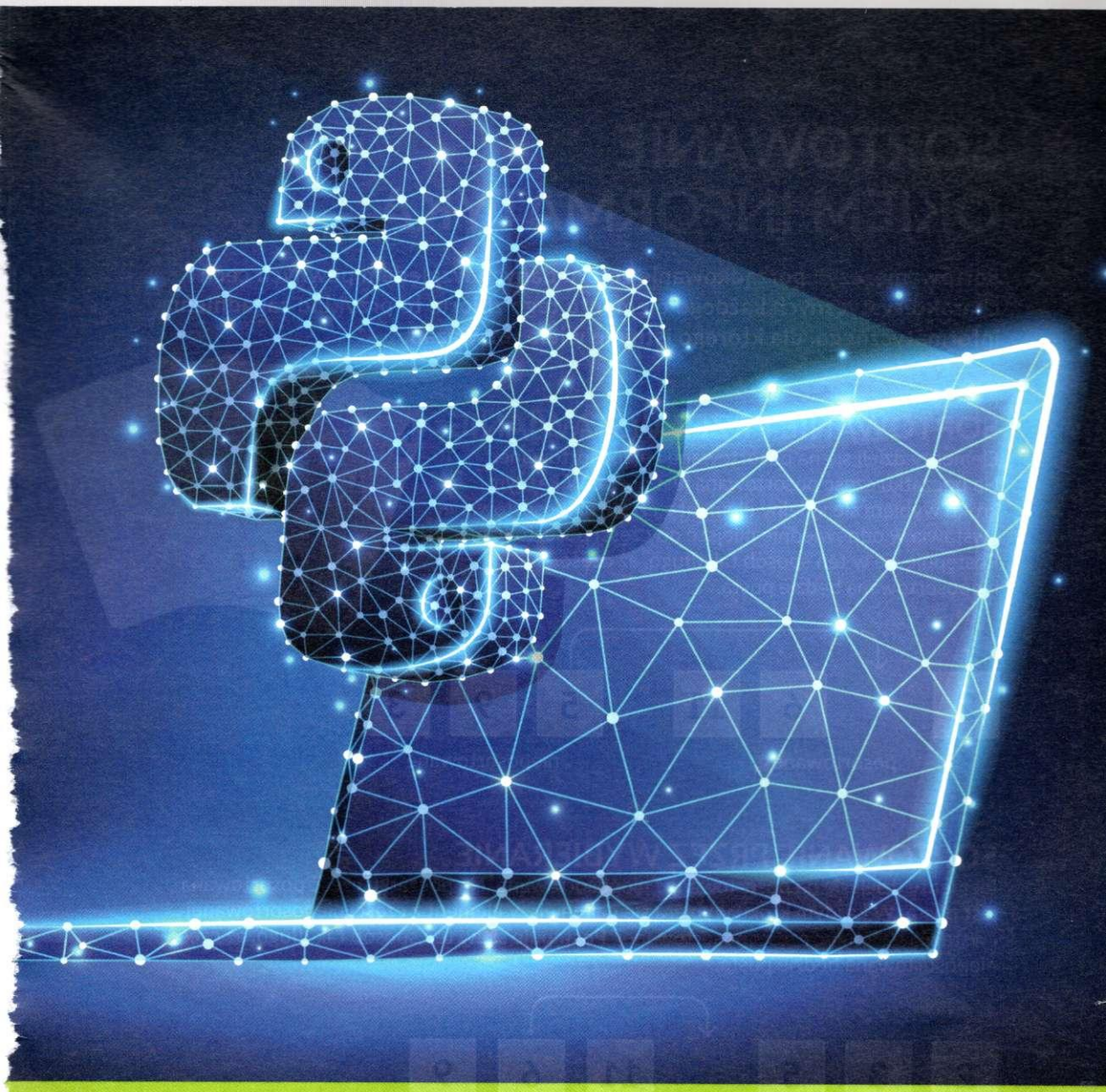


ALGORYTMIKA I PROGRAMOWANIE W PYTHONIE

- Zastosowania algorytmu Euklidesa
- Złożoność obliczeniowa algorytmu
- Metody sortowania
- Przykłady algorytmów zachłannych

SORTOWANIE OKIEM INFORMATYKA

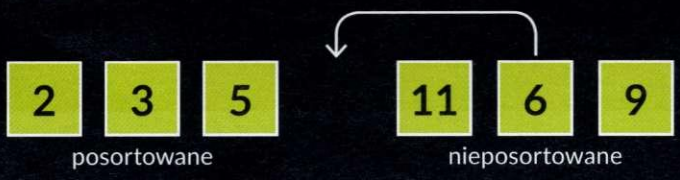
Sortowanie, czyli porządkowanie zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru, to przykład zagadnienia informatycznego, dla którego istnieje wiele rozwiązań.

SORTOWANIE PRZEZ WSTAWIANIE

Elementy dzieli się na posortowane i nieposortowane. Na początku część posortowana jest pusta. W każdym kroku wybiera się kolejny element z części nieposortowanej i wstawia w odpowiednie miejsce do części posortowanej. Postępuje się w ten sposób tak długo, aż część nieposortowana będzie pusta.

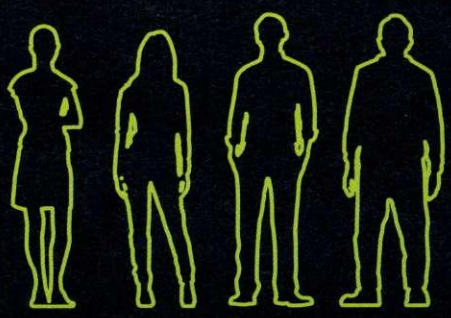
SORTOWANIE PRZEZ WYBIERANIE

Elementy dzieli się na posortowane i nieposortowane. Na początku część posortowana jest pusta. W każdym kroku wybiera się najmniejszy element z części nieposortowanej i wstawia na koniec części posortowanej. Postępuje się w ten sposób tak długo, aż część nieposortowana będzie pusta.



SORTOWANIE BĄBELKOWE

Rozpatruje się elementy parami, przechodząc od lewej do prawej. Porównuje się dwa sąsiednie elementy i odpowiednio zamienia kolejność. Postępuje się w ten sposób tak długo, aż do przejścia bez żadnej zmiany.



SORTOWANIE PRZEZ ZLICZANIE

Przygotowuje się uporządkowaną rosnąco listę występujących elementów i ustawia licznik wystąpienia każdego elementu na 0. Przegląda się kolejno elementy zbioru i zlicza ich wystąpienia. Na koniec wypisuje się elementy według liczby wystąpień.

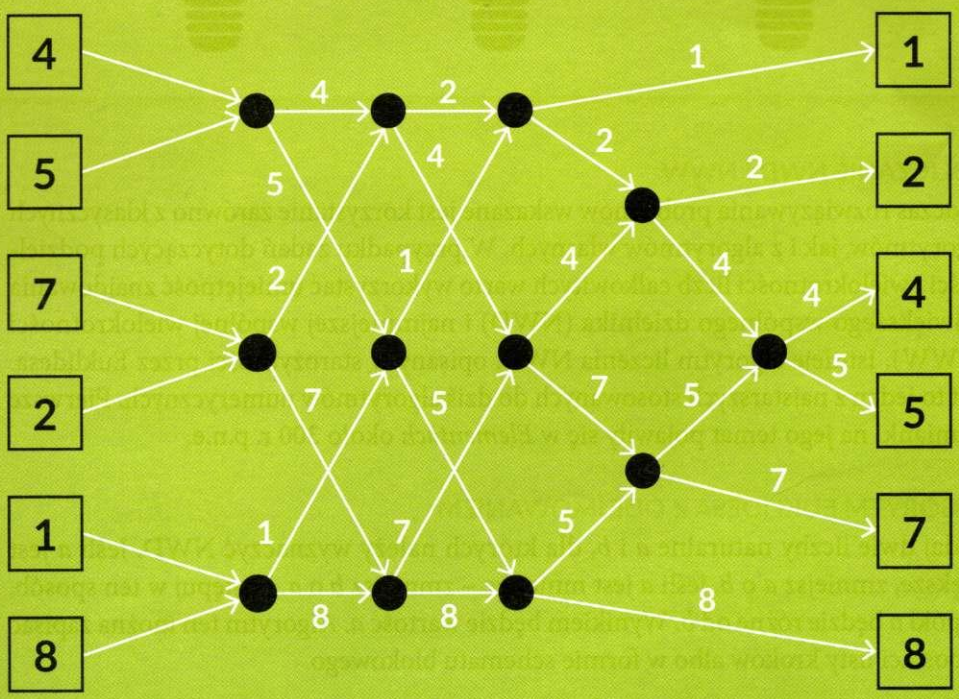
4	2	2	3	1	6	6	4	3	3	5	1	2	5	3	4

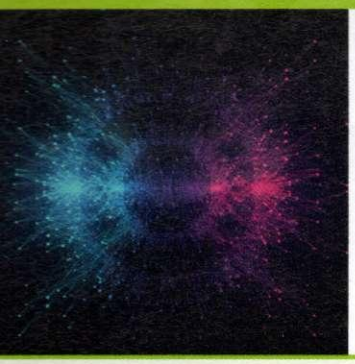
ELEMENT	1	2	3	4	5	6
ILE RAZY WYSTĘPUJE	2	3	4	3	2	2

1	1	2	2	2	3	3	3	3	4	4	4	5	5	6	6

SORTOWANIE RÓWNOLEGŁE

Porównuje się elementy na wejściu i wyjściu sieci sortującej – przekazuje się mniejszy element na górze, a większy na dole. Na koniec otrzymuje się posortowany ciąg.





2. Algorytm Euklidesa w praktyce

- Pętla warunkowa `while`
- Stosowanie algorytmu Euklidesa do rozwiązywania zadań
- Działania na ułamkach z wykorzystaniem NWD i NWW

ZADANIE NA START

Po ilu sekundach lampki zamigają jednocześnie

Klasa 2c będzie gościć uczniów z innego kraju w ramach wymiany międzynarodowej. Uczniowie przygotowują neon powitalny, na którym migają trzy rodzaje lampek. Wszystkie lampki właśnie zamigały jednocześnie. Jedna grupa lampek miga co a sekund, druga – co b sekund, a trzecia – co c sekund. Po ilu sekundach wszystkie lampki znów zamigają jednocześnie dla $a = 2$, $b = 3$ i $c = 4$? A dla $a = 24$, $b = 36$ i $c = 60$? Czy potrafisz zapisać wzór ogólny?

OBLICZANIE NWD I NWW

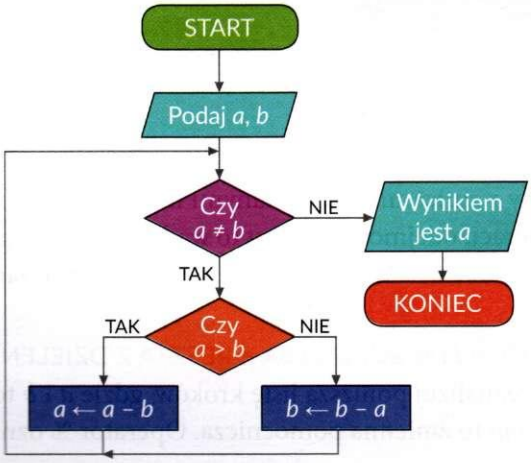
Podczas rozwiązywania problemów wskazane jest korzystanie zarówno z klasycznych algorytmów, jak i z algorytmów własnych. W przypadku zadań dotyczących podzielności i wielokrotności liczb całkowitych warto wykorzystać umiejętność znajdowania największego wspólnego dzielnika (NWD) i najmniejszej wspólnej wielokrotności (NWW). Istnieje algorytm liczenia NWD, opisany w starożytności przez Euklidesa. Jest to jeden z najstarszych stosowanych do dziś algorytmów numerycznych. Pierwsze wzmianki na jego temat pojawiły się w *Elementach* około 300 r. p.n.e.

ALGORYTM EUKLIDESA Z ODEJMOWANIEM

Podaj dwie liczby naturalne a i b, dla których należy wyznaczyć NWD. Jeśli a jest większe, zmniejsz a o b, jeśli a jest mniejsze – zmniejsz b o a. Postępuj w ten sposób, dopóki a będzie różne od b. Wynikiem będzie wartość a. Algorytm ten można zapisać w postaci listy kroków albo w formie schematu blokowego.

Zauważ, że w każdym kroku zmieniasz większą liczbę na różnicę badanych liczb i pozostawiasz mniejszą liczbę bez zmian.

1. Podaj a i b.
2. Dopóki $a \neq b$, wykonuj:
   a. jeśli $a > b$, to $a \leftarrow a - b$;
   b. jeśli $a < b$, to $b \leftarrow b - a$.
3. Wynikiem jest a.



Rys. 1. Algorytm Euklidesa w wersji z odejmowaniem – lista kroków i schemat blokowy

Przeanalizuj to na przykładzie. Oblicz największe wspólne dzielniki liczb 25 i 35 oraz 39 i 12.

a	b
25	35
25	10
15	10
5	10
5	5

a	b
39	12
27	12
15	12
3	12
3	9
3	6
3	3

A teraz przeanalizuj zapis w Pythonie.

```
1. def nwd(a, b):
2.     while a != b:
3.         if a > b:
4.             a = a - b
5.         else:
6.             b = b - a
7.     return a
```

Rys. 2. Funkcja realizująca algorytm Euklidesa w wersji z odejmowaniem

W funkcji użyto pętli warunkowej **while** – dopóki warunek jest spełniony, wykonywane są instrukcje w pętli. W tym przypadku pętla zakończy działanie, gdy *a* jest równe *b*.

ZAPAMIĘTAJ!

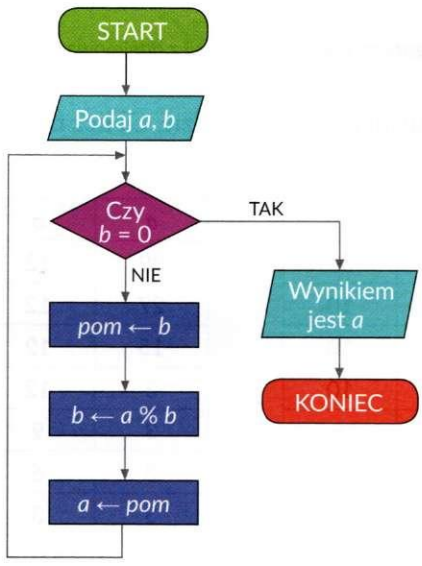
```
while <wyrażenie jest prawdziwe>:
    <instrukcje>
```

Gdy różnica między badanymi liczbami jest znacząca, należy wykonać dużą liczbę operacji odejmowania. Jest to wada tej wersji algorytmu. W drugim algorytmie odejmowanie zostało zastąpione operacją wyznaczania reszty z dzielenia.

ALGORYTM EUKLIDESA Z RESZTĄ Z DZIELENIA

Przeanalizuj poniższą listę kroków, gdzie *a* i *b* to liczby, dla których szukamy NWD, a *pom* to zmienna pomocnicza. Operator % oznacza resztę z dzielenia.

1. Podaj *a* i *b*.
2. Dopóki $b \neq 0$, wykonuj:
   a. $pom \leftarrow b$;
   b. $b \leftarrow a \% b$;
   c. $a \leftarrow pom$.
3. Wynikiem jest *a*.



Rys. 3. Algorytm Euklidesa w wersji z resztą z dzielenia – lista kroków i schemat blokowy

A teraz jeszcze raz wyznacz NWD dla liczb 25 i 35 oraz 39 i 12.

a	*b*	*pom*
25	35	
35	25	35
25	10	25
10	5	10
5	0	5

a	*b*	*pom*
39	12	
12	3	12
3	0	3

I przeanalizuj zapis w języku Python.

```
1. def nwd(a, b):
2.     while b != 0:
3.         pom = b
4.         b = a % b
5.         a = pom
6.     return a
```

Rys. 4. Funkcja realizująca algorytm Euklidesa w wersji z resztą z dzielenia

WIEM WIĘCEJ

Moduły to pliki z biblioteki standardowej Pythona zawierające pewien zestaw funkcji gotowych do wykorzystania. W module **math** jest zaimplementowana dwuargumentowa funkcja **gcd**, która wyznacza NWD. Nazwa funkcji nawiązuje do angielskiej nazwy największego wspólnego dzielnika – *greatest common divisor*.

Ćwiczenie 1. Podział na rzędy

W spotkaniu weźmie udział określona liczba gości z różnych krajów i określona liczba gospodarzy. Zaproszeni będą siedzieć w rzędach tak, aby każdy rząd był równoliczny i w każdym siedzieli albo tylko goście, albo tylko gospodarze. Jaka jest największa możliwa liczba osób w każdym rzędzie? Przeanalizuj dane w tabeli i zdefiniuj funkcję **rzad(goscie, gospodarze)**, której parametrami są liczba gości i gospodarzy, a wynikiem jest liczebność rzędu.

Wywołanie funkcji	Wynik
`rzad(72, 90)`	18
`rzad(24, 36)`	12

- Problem sprowadza się do obliczenia NWD liczb podanych jako parametr. Każdy rząd ma się składać z takiej samej liczby osób – wynik musi więc być dzielnikiem obu parametrów i być możliwie największy.
- Zdefiniuj funkcję pomocniczą **nwd(a, b)** i funkcję **rzad(goscie, gospodarze)**.

```
1. def nwd(a, b):
2.     while b != 0:
3.         pom = b
4.         b = a % b
5.         a = pom
6.     return a
7.
8. def rzad(goscie, gospodarze):
9.     return nwd(goscie, gospodarze)
```

- Sprawdź działanie skryptu z różnymi danymi.

```
10. print(rzad(72,90))
11. print(rzad(24,36))
```

Pamiętaj, aby następne ćwiczenia w tej lekcji wykonywać w tym samym pliku.

Ćwiczenie 2. Podział na grupy

Należy podzielić gości i gospodarzy na możliwie małe drużyny tak, by w każdej drużynie była jednakowa liczba osób (gości i gospodarzy). Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile(goscie, gospodarze)**, której parametrami są liczba gości oraz liczba gospodarzy, a wynikiem jest liczebność każdej drużyny.

Wywołanie funkcji	Wynik
ile(25, 35)	**12**
ile(49, 42)	**13**

- Przeanalizuj problem na podstawie danych z tabeli. Aby określić liczebność drużyny, należy obliczyć NWD, a potem zsumować liczbę osób w jednej grupie.

  NWD(25, 35) = 5

  25 : 5 + 35 : 5 = 5 + 7 = 12

  W pierwszym przypadku drużyny są złożone z 5 gości + 7 gospodarzy = 12 osób. Zwróć uwagę, że musisz operować na liczbach całkowitych, wobec czego konieczne jest zastosowanie dzielenia całkowitego.

- Zdefiniuj funkcję **ile(goscie, gospodarze)** zawierającą zmienną pomocniczą, która będzie przechowywać wartości NWD.

```
1. def ile(goscie, gospodarze):
2.     pom = nwd(goscie, gospodarze)
3.     return goscie // pom + gospodarze // pom
```

- Sprawdź działanie skryptu z różnymi danymi.

NWW

Wspólna wielokrotność liczb naturalnych a i b to każda liczba naturalna, która dzieli się przez a oraz przez b. Jeśli chcesz znaleźć najmniejszą taką liczbę, możesz skorzystać ze wzoru $\mathrm{NWW}(a, b) = (a \cdot b) : \mathrm{NWD}(a, b)$. Czy umiesz to zapisać w Pythonie?

```
1. def nww(a, b):
2.     return a * b // nwd(a, b)
```

Rys. 5. Wyznaczanie NWW na podstawie NWD

Ćwiczenie 3. Spotkania zespołów

W ramach przygotowań do spotkania powstały dwa zespoły: zespół programowy i zespół przygotowujący materiały. Pierwszy spotyka się co **x** dni, drugi co **y**. Dziś spotkały się oba zespoły. Przeanalizuj dane w tabeli i zdefiniuj funkcję **kiedy(x, y)**, w której jako parametry będziesz podawać, co ile dni spotykają się zespoły, a wynikiem będzie liczba dni, jakie upłyną do następnego wspólnego spotkania obu zespołów.

Wywołanie funkcji	Wynik
kiedy(6, 4)	**12**
kiedy(9, 12)	**36**

- Problem wymaga wyznaczenia wartości NWW na podstawie wartości NWD. Skorzystaj ze wzoru NWW(*a*, *b*) = (*a* · *b*) : NWD(*a*, *b*). Dla danych z tabeli obliczenia przedstawiają się następująco: NWD(6, 4) = 2; NWW(6, 4) = (6 · 4) : 2 = 12.
- Zdefiniuj funkcję **nww(x, y)**, a następnie funkcję **kiedy(x, y)**.

```
1. def kiedy(x, y):
2.     return nww(x, y)
```

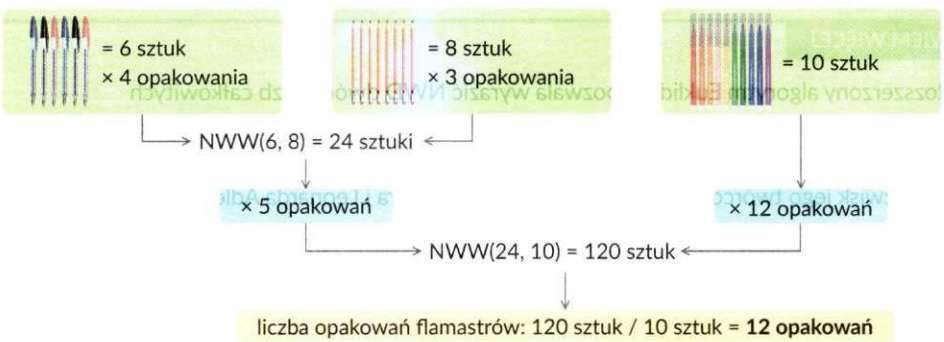
- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 4. Sprytne zakupy

Każdy gość powinien dostać taką samą liczbę długopisów, ołówków i flamastrów. Przeanalizuj dane w tabeli i zdefiniuj funkcję **flamastry(x, y, z)**, której parametrami są liczba długopisów w paczce **(x)**, liczba ołówków w paczce **(y)** i liczba sztuk flamastrów w paczce **(z)**, a wynikiem jest najmniejsza liczba opakowań flamastrów, jakie gospodarze powinni kupić, aby nic nie zostało.

Wywołanie funkcji	Wynik
flamastry(6, 8, 10)	**12**
flamastry(39, 26, 52)	**3**

- Przeanalizuj problem dla danych z tabeli.



- Aby rozwiązać problem, należy znaleźć NWW trzech parametrów i podzielić otrzymaną wartość przez ostatni parametr. W tym celu można najpierw wyznaczyć NWW dla **x** i **y** i oznaczyć tę wartość jako **pom1**, a następnie wyznaczyć NWW dla **pom1** i **z**, czyli dla wszystkich trzech parametrów, i oznaczyć tę wartość jako **pom2**. Potem wystarczy podzielić **pom2** przez **z**.
- Zdefiniuj funkcję **flamastry(x, y, z)**.

```
1. def flamastry(x, y, z):
2.     pom1 = nww(x, y)
3.     pom2 = nww(pom1, z)
4.     return pom2 // z
```

- Sprawdź działanie skryptu z różnymi danymi.

NWW I NWD A UŁAMKI ZWYKŁE

Algorytm dodawania i odejmowania ułamków zwykłych można opisać z wykorzystaniem NWD i NWW.

1. Podaj licznik i mianownik pierwszego ułamka: a oraz b.
2. Podaj licznik i mianownik drugiego ułamka: c oraz d.
3. Znajdź wspólny mianownik m = NWW(b, d).
4. Rozszerz licznik pierwszego ułamka $a2 = a * m // b$.
5. Rozszerz licznik drugiego ułamka $c2 = c * m // d$.
6. Dodaj do siebie liczniki $l = a2 + c2$.
7. Skróć licznik i mianownik: $l = l$ // nwd(l, m) oraz $m = m$ // nwd(l, m).
8. Wyodrębnij całości: $x = l // m$ oraz $l = l \% m$.
9. Wyniki to całość x, licznik l i mianownik m.

Ćwiczenie 5. Mnożenie ułamków

Przedstaw pisemnie algorytm mnożenia dwóch ułamków zwykłych.

Ćwiczenie 6. Dzielenie ułamków

Przedstaw pisemnie algorytm dzielenia dwóch ułamków zwykłych.

WIEM WIĘCEJ

Rozszerzony algorytm Euklidesa pozwala wyrazić NWD dwóch liczb całkowitych dodatnich a i b w postaci wzoru NWD(a, b) = $a \cdot x + b \cdot y$, gdzie x, y to liczby całkowite. Znajduje on zastosowanie w algorytmie kryptograficznym RSA, którego nazwa nawiązuje do nazwisk jego twórców: Rona Rivesta, Adiego Shamira i Leonarda Adlemana.

PODSUMOWANIE

- Podczas rozwiązywania zadań wskazane jest korzystanie zarówno z klasycznych algorytmów, jak i z algorytmów własnych.
- Jedną z ważnych umiejętności programistycznych jest umiejętność zapisu algorytmów znanych z życia codziennego językiem formalnym.
- Pętla warunkowa jest wykonywana w zależności od podanego warunku. W przypadku pętli **`while`** zawarte w niej instrukcje są wykonywane, dopóki warunek jest spełniony.
- W życiu codziennym występuje wiele problemów, które można rozwiązać z wykorzystaniem umiejętności znalezienia NWD i NWW. W matematyce ta umiejętność przydaje się przy rozwiązywaniu zadań na ułamkach zwykłych, a rozszerzony algorytm Euklidesa ma zastosowanie w szyfrowaniu.

PYTANIA SPRAWDZAJĄCE

1. Jakie znasz algorytmy obliczania NWD?
2. Dlaczego przy rozwiązywaniu zadań stosuje się zmienne pomocnicze?
3. Kiedy warto wyznaczyć NWD dwóch liczb? Podaj przykłady takich problemów.

ZADANIA DODATKOWE

Zadanie 1. Sprawdź się!

W Akademii Khana (**https://pl-pl.khanacademy.org**) można znaleźć ciekawe zadania tekstowe dotyczące NWD i NWW. Spróbuj je rozwiązać.

Zadanie 2. Implementacja NWD i NWW

Zaimplementuj w języku Python dodawanie, odejmowanie, mnożenie i dzielenie dwóch ułamków zwykłych. Wykorzystaj algorytmy NWD i NWW.

3. Badanie własności liczb całkowitych

- Sprawdzanie, czy liczba jest pierwsza, czy złożona
- Porównywanie i ocena algorytmów
- Badanie szczególnych własności liczb całkowitych

ZADANIE NA START

Sito Eratostenesa

Liczby od 2 do 100 wypisano w kratkach, a następnie kratkę z liczbą 2 oznaczono kolorem zielonym i wykreślono wszystkie jej wielokrotności. Potem wybrano pierwszą liczbę niewykreśloną, czyli 3, jej kratkę oznaczono kolorem zielonym i podobnie jak poprzednio wykreślono wszystkie jej wielokrotności. Proces ten kontynuowano, aż znaleziono wszystkie liczby pierwsze. Jak długo trzeba badać wielokrotności liczb pierwszych, aby mieć pewność, że zostały wykreślone wszystkie liczby złożone, w przypadku zbioru liczb od 2 do 100 i od 2 do n?

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

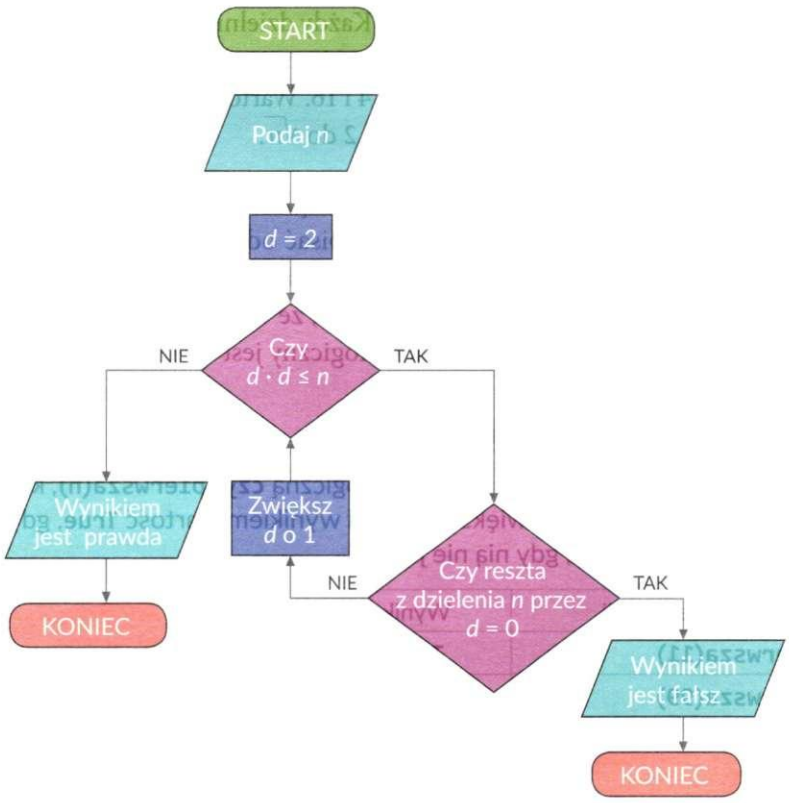
LICZBY PIERWSZE

Liczba pierwsza to taka liczba naturalna, która ma dokładnie dwa dzielniki naturalne: jedynkę i samą siebie. Z kolei liczba złożona to liczba naturalna większa od 1, mająca co najmniej jeden naturalny dzielnik różny od 1 i od niej samej. Jak sprawdzić, czy dana liczba jest pierwsza?

1. Wczytaj liczbę $n > 1$.
2. Przypisz zmiennej d wartość 2.
3. Sprawdź, czy jest spełniony warunek, że $d \cdot d \leq n$.

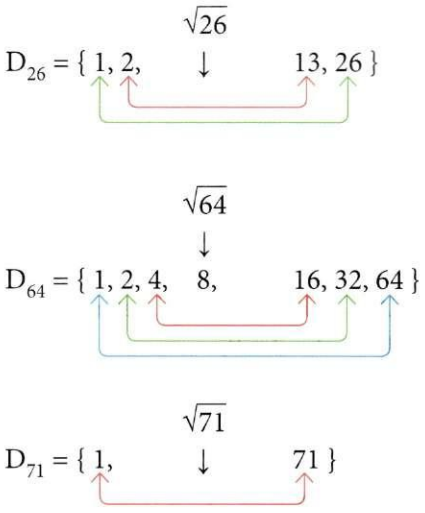
- Jeśli odpowiedź brzmi NIE, liczba jest liczbą pierwszą – należy zakończyć działanie algorytmu, np. $n = 3$, $d = 2$; $2 \cdot 2 \leq 3 \rightarrow$ NIE $\rightarrow$ Liczba 3 jest liczbą pierwszą.
- Jeśli odpowiedź brzmi TAK, oblicz resztę z dzielenia liczby n przez zmienną d, np. $n = 4$, $d = 2$; $2 \cdot 2 \leq 4$; $4 : 2 = 2$ r. 0;
  np. $n = 5$, $d = 2$; $2 \cdot 2 \leq 5$; $5 : 2 = 2$ r. 1.
  - ✓ Jeśli reszta wynosi 0, zakończ działanie algorytmu – istnieje przynajmniej jeden dzielnik liczby, więc nie jest to liczba pierwsza.
  - ✓ Jeśli reszta jest różna od 0, wskaż nowy dzielnik (zwiększ wartość zmiennej d o 1) i wróć do punktu 3. Jeśli nie znajdziesz żadnego dzielnika, to badana liczba jest liczbą pierwszą, np. $n = 5$, $d = 2 + 1 = 3$; $3 \cdot 3 \leq 5 \rightarrow$ NIE $\rightarrow$ Liczba 5 jest liczbą pierwszą.

Przeanalizuj schemat blokowy opisanego algorytmu.



Rys. 1. Schemat blokowy algorytmu – czy dana liczba jest liczbą pierwszą

Przeanalizuj poniższe przykłady dla liczb: 26, 64 i 71. Jak wśród dzielników umieszczono wartość $\sqrt{n}$?



Rys. 2. Dzielniki wybranych liczb

Jeżeli liczba ma dzielniki większe od 1 i mniejsze od niej samej, to jest ich tyle samo po lewej i po prawej stronie pierwiastka z liczby. Każdy dzielnik niebędący pierwiastkiem ma swoją parę. Jeżeli d jest dzielnikiem liczby, to jest nim również iloraz n przez d, np. parami dzielników dla liczby 64 są 2 i 32 oraz 4 i 16. Warto zauważyć, że potencjalnych dzielników wystarczy szukać w przedziale od 2 do $\sqrt{n}$.

Tego typu własności liczb warto badać za pomocą komputera, szczególnie gdy analizowane są liczby wielocyfrowe. Wystarczy zapisać odpowiedni algorytm i pozwolić komputerowi wykonać kolejne operacje. Jak zrealizować algorytm w środowisku Python? Analiza schematu blokowego pokazuje, że warto zastosować pętlę **`while`**, która wykonuje się, dopóki pewien warunek logiczny jest spełniony.

Ćwiczenie 1. Czy liczba jest pierwsza?

Przeanalizuj dane w tabeli i zdefiniuj funkcję logiczną **`czy_pierwsza(n)`**, której parametrem jest liczba naturalna **`n`** większa od 1, a wynikiem wartość **`True`**, gdy jest ona liczbą pierwszą, albo **`False`**, gdy nią nie jest.

Wywołanie funkcji	Wynik
`czy_pierwsza(11)`	`True`
`czy_pierwsza(99)`	`False`

- Problem sprowadza się do zapisania omówionego wcześniej algorytmu językiem formalnym. Należy sprawdzać podzielność liczby przez kolejne dzielniki (począwszy od 2), dopóki iloczyn dzielnik razy dzielnik jest mniejszy lub równy liczbie. Jeśli zostanie znaleziony pierwszy dzielnik liczby, funkcja powinna zakończyć działanie wynikiem **False**. Jeśli nie znajdzie się żaden dzielnik, funkcja powinna zakończyć działanie wynikiem **True**.
- Zdefiniuj funkcję **czy_pierwsza(n)**.

```
1. def czy_pierwsza(n):
2.     d = 2
3.     while d * d <= n:
4.         if n % d == 0:
5.             return False
6.         d += 1
7.     return True
```

- Sprawdź działanie funkcji dla różnych danych. Wybierz kilka większych liczb pierwszych, np. 541, 7919, 48 611, oraz parę liczb niebędących pierwszymi, np. 679, 9231, 76 613.

WIEM WIĘCEJ

Istnieją liczby pierwsze złożone z kolejnych cyfr: 23, 67, 4567, 23456789, 1234567891.
Liczba 11111111111111111111 złożona z 23 jedynek także jest pierwsza.
Liczba 73939133 jest pierwsza, ponadto liczby otrzymane przez usuwanie z niej cyfr od prawej też są pierwsze: 7393913, 739391, 73939, 7393, 739, 73, 7.

Ćwiczenie 2. N-ta liczba pierwsza

Przeanalizuj dane w tabeli i zdefiniuj funkcję **pierwsza(n)**, której parametrem będzie liczba naturalna **n**, a wynikiem – *n*-ta liczba pierwsza.

Wywołanie funkcji	Wynik
pierwsza(7)	**17**
pierwsza(25)	**97**

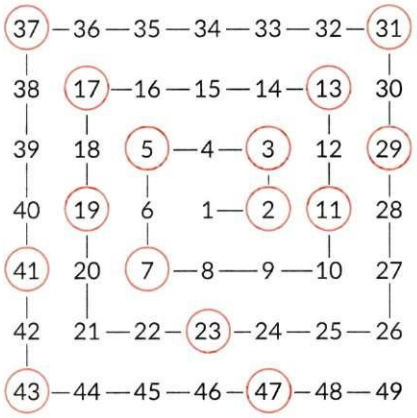
- Przyjrzyj się rysunkowi zamieszczonemu w zadaniu na start. Znajdź siódmą i dwudziestą piątą liczbę pierwszą. W jaki sposób będziesz postępować?
  Możesz sprawdzić kolejne liczby, czy są pierwsze, i zliczać te, które nimi są. *N*-ta liczba pierwsza będzie wynikiem. Jeśli nie natrafisz na liczbę pierwszą, będziesz zwiększać wartość zmiennej **numer**; jeśli aktualny numer będzie równy parametrowi, to znaczy, że ostatnia znaleziona liczba pierwsza jest wynikiem.
- Skorzystaj z rozwiązania poprzedniego ćwiczenia i rozszerz je o definicję funkcji **pierwsza(n)**.

```
1. def pierwsza(n):
2.     numer = 0
3.     i = 2
4.     while numer != n:
5.         if czy_pierwsza(i):
6.             numer += 1
7.         i += 1
8.     return i - 1
```

- Sprawdź działanie funkcji dla różnych danych. Na portalu **https://www.matemaks.pl/liczby-pierwsze.html** znajdziesz generator liczb pierwszych. Porównaj wyniki działania swojej funkcji dla **n** = 100, **n** = 870 i **n** = 10 000.

WIEM WIĘCEJ

Spirala Ulama to graficzna metoda przedstawiania pewnych prawidłowości w rozkładzie liczb pierwszych. Kolejne liczby naturalne są zapisane spiralnie. Widać, że na niektórych przekątnych liczby pierwsze grupują się częściej niż na innych. Fakt ten nie został do tej pory wyjaśniony. To zjawisko występuje także, jeśli na początku są inne wartości niż 1.



Ćwiczenie 3. Liczby bliźniacze

Przyjrzyj się liczbom pierwszym zaznaczonym na rysunku w zadaniu na start. Zauważ, że niektóre z nich różnią się tylko o 2. Takie liczby nazywają się bliźniaczymi. W pierwszej setce jest osiem takich par liczb: 3 i 5, 5 i 7, 11 i 13, 17 i 19, 29 i 31, 41 i 43, 59 i 61, 71 i 73. Przeanalizuj dane w tabeli i zdefiniuj funkcję **blizniacze(n)**, której parametrem jest liczba naturalna **n**, a wynikiem pierwsza liczba z *n*-tej pary liczb bliźniaczych.

Wywołanie funkcji	Wynik
blizniacze(3)	**11**
blizniacze(7)	**59**

ZŁOŻONOŚĆ OBLICZENIOWA ALGORYTMU

Algorytm jest poprawny, jeśli daje poprawny wynik w skończonym czasie, a dla poprawnych danych wejściowych zawsze się kończy. Oprócz tego powinien działać szybko i nie zajmować zbyt dużo miejsca.

Złożoność czasowa ułatwia oszacowanie czasu działania algorytmu, tj. czasu niezbędnego do rozwiązania problemu w zależności od rozmiaru danych wejściowych. Najczęściej nie mierzy się fizycznie czasu działania algorytmu, ale rozpatruje się operacje dominujące, czyli operacje, które są wykonywane podczas działania algorytmu i determinują czas jego działania – może to być dodawanie, mnożenie, porównywanie i przestawianie elementów itp. W algorytmie przedstawionym na rys. 3 operacja dominująca znajduje się w wierszu 4. Operacja ta wykona się tyle razy, ile wynosi rozmiar danych, tj. **n**.

```
1. n = 20
2. w = 0
3. for i in range(n):
4.     w = w + i
5. print(w)
```

Rys. 3. Algorytm o złożoności czasowej liniowej – liczba operacji dominujących, które należy wykonać, aby otrzymać rozwiązanie problemu, wynosi **n**

Podczas badania, czy dana liczba jest liczbą pierwszą, ogranicza się liczbę operacji do $\sqrt{n}$, np. dla $n = 1\ 000\ 000$ wystarczy sprawdzać podzielność tylko do 1000.

Złożoność pamięciowa określa liczbę komórek pamięci, która będzie zajęta w trakcie wykonywania algorytmu. Ta złożoność jest najczęściej proporcjonalna do liczby zmiennych użytych w algorytmie.

ZAPAMIĘTAJ!

Aby obliczyć pierwiastek kwadratowy z liczby, można skorzystać z modułu **math** i za pomocą polecenia **import** importować do programu funkcję **sqrt** – na początku skryptu należy wówczas wpisać instrukcję **from math import sqrt**.

Ćwiczenie 4. Suma dzielników liczby

Przeanalizuj dane w tabeli i zdefiniuj funkcję **suma_dzielnikow(n)**, której parametrem jest liczba naturalna **n**, a wynikiem – suma dzielników tej liczby. Sformułuj dwa algorytmy i porównaj szybkość działania każdego z nich dla różnych danych.

Wywołanie funkcji	Wynik
suma_dzielnikow(7)	**8**
suma_dzielnikow(25)	**31**

- Pierwszy algorytm będzie sprawdzał podzielność danej liczby przez kolejne dzielniki od 1 (najmniejszy dzielnik) do niej samej (największy dzielnik).

```
1. def suma_dzielnikow_1(n):
2.     suma = 0
3.     for i in range(1, n + 1):
4.         if n % i == 0:
5.             suma += i
6.     return suma
```

- Drugi algorytm będzie sprawdzał podzielność danej liczby przez kolejne dzielniki od 2 do pierwiastka z liczby. Jeśli znajdzie dzielnik, to doda również jego parę. W przypadku gdy pierwiastek z liczby jest liczbą naturalną, jest on też dzielnikiem danej liczby. Wartość początkowa jest równa sumie dzielników oczywistych (1 i liczba).

```
1. from math import sqrt
2.
3. def suma_dzielnikow_2(n):
4.     suma = n + 1
5.     for i in range(2, int(sqrt(n)) + 1):
6.         if n % i == 0:
7.             suma += i
8.             if i != n // i:
9.                 suma += n // i
10.    return suma
```

- Porównaj poprawność wyników i czas działania obu algorytmów.
- Oba rozwiązania warto przetestować dla następujących liczb: 6, 28, 496, 8128, 33 550 336, 8 589 869 056. Są to tzw. **liczby doskonałe**, czyli takie, które są sumą wszystkich swoich dzielników właściwych (to znaczy mniejszych od nich). Podwojona wartość liczby doskonałej będzie sumą wszystkich jej dzielników. Dla piątej liczby doskonałej (ośmiocyfrowej) widać różnicę czasu działania algorytmów. Dla szóstej liczby doskonałej (dziesięciocyfrowej) w przypadku pierwszego algorytmu trudno doczekać się wyniku.

Ćwiczenie 5. Liczby zaprzyjaźnione

Liczby zaprzyjaźnione to dwie liczby naturalne, z których każda jest równa sumie dzielników właściwych drugiej liczby, np. 220 i 284, 1184 i 1210, 2620 i 2924. Przeanalizuj dane w tabeli i zdefiniuj funkcję **zaprzyjaznione(n)**, której parametrem będzie liczba naturalna **n**, a wynikiem – mniejsza liczba z *n*-tej pary liczb zaprzyjaźnionych.

Wywołanie funkcji	Wynik
zaprzyjaznione(1)	**220**
zaprzyjaznione(3)	**2620**

WIEM WIĘCEJ

Dzięki liczbom pierwszym możemy czuć się bezpiecznie. Poczta elektroniczna, operacje między bankami czy przesyłane dane są szyfrowane na podstawie własności liczb pierwszych. Działanie algorytmu RSA z kluczem publicznym jest oparte na trudności rozkładu dużych liczb na czynniki pierwsze. O ile stosunkowo łatwo jest pomnożyć dwie liczby przez siebie, o tyle operacja odwrotna – rozkład liczby na czynniki pierwsze – jest czasochłonna.

PODSUMOWANIE

- Większość problemów algorytmicznych można rozwiązać na kilka sposobów. Warto w takim przypadku zwracać uwagę na złożoność obliczeniową, aby wybrać optymalne rozwiązanie.
- Program komputerowy rozwiązujący problem ma do dyspozycji dwa podstawowe zasoby: czas i pamięć.
- Czasowa złożoność obliczeniowa określa czas działania algorytmu rozumianego jako czas niezbędny do rozwiązania problemu w zależności od rozmiaru danych wejściowych. W ten sposób uzyskany wynik jest niezależny od faktycznej szybkości komputera.
- Operacje dominujące to operacje, które należy wykonać, aby otrzymać rozwiązanie problemu. Determinują one czas działania algorytmu.
- Pamięciowa złożoność obliczeniowa określa liczbę komórek pamięci zajętych w trakcie wykonywania algorytmu.
- Złożoność pamięciowa jest zwykle proporcjonalna do liczby zmiennych w algorytmie.
- Funkcję `sqrt()` obliczania pierwiastka kwadratowego z liczby należy importować z modułu `math`.

PYTANIA SPRAWDZAJĄCE

1. Jak badać podzielność liczby, aby sprawdzić, czy jest ona pierwsza?
2. Kiedy algorytm jest poprawny?
3. Od czego zależy złożoność obliczeniowa algorytmu?

ZADANIA DODATKOWE

Zadanie 1. Liczby czworacze

Liczby czworacze to liczby pierwsze mające postać: n, $n + 2$, $n + 6$ i $n + 8$, np. 5, 7, 11, 13 i 11, 13, 17, 19 i 101, 103, 107, 109. Przeanalizuj dane w tabeli i zdefiniuj funkcję **czworacze(n)**, której parametrem będzie liczba naturalna **n**, a wynikiem – lista takich liczb czworaczych, gdzie pierwsza liczba będzie większa od podanego parametru.

Wywołanie funkcji	Wynik
`czworacze(9)`	`[11, 13, 17, 19]`
`czworacze(150)`	`[191, 193, 197, 199]`

Zadanie 2. Problem Collatza

Ciąg liczb Collatza zdefiniowany jest następująco: pierwsza liczba ciągu jest dowolną liczbą naturalną x, a każda kolejna wartość ciągu obliczana jest na podstawie poprzedniej według poniższych zasad:

- jeśli poprzednia wartość była parzysta, to należy podzielić ją przez 2;
- jeśli poprzednia wartość była nieparzysta, to należy pomnożyć ją przez 3 i dodać 1.

Wobec tego dla wartości początkowej $x = 10$ kolejne liczby to: 5, 16, 8, 4, 2, 1.
Przeanalizuj dane w tabeli i zdefiniuj funkcję **collatz(x)**, której parametrem będzie liczba naturalna **x**, czyli wartość początkowa ciągu liczb Collatza, a wynikiem – liczba kroków, po których w ciągu pojawi się liczba 1.

Wywołanie funkcji	Wynik
`collatz(10)`	`6`
`collatz(15)`	`17`

4. Sortowanie bąbelkowe i przez wstawianie

- Sortowanie danych
- Sortowanie metodą bąbelkową
- Sortowanie przez wstawianie

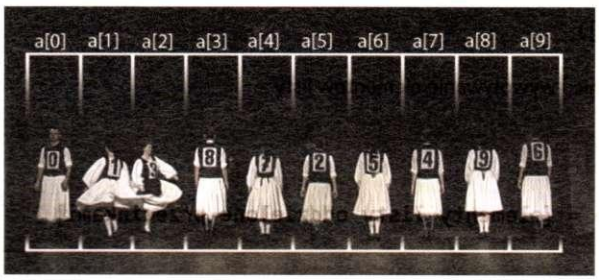
ZADANIE NA START

Taneczne porządkowanie

Obejrzyj wizualizację jednego z algorytmów sortowania przedstawionego z wykorzystaniem tańca:

- węgierskiego – **https://www.youtube.com/watch?v=lyZQPjUT5B4**;
- rumuńskiego – **https://www.youtube.com/watch?v=ROalU379l3U**.

Opisz swoimi słowami, na czym polega algorytm.



SORTOWANIE DANYCH

Sortowanie to zadanie polegające na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych, jakiegoś klucza. Jeden zbiór danych można różnie sortować w zależności od przyjętego kryterium i porządku, np.

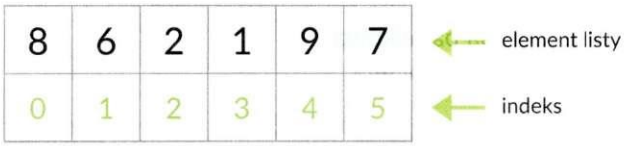
- karty do gry – według kolorów i wartości, w zależności od zasad danej gry;
- spis książek w bibliotece – alfabetycznie według autorów lub tytułów;
- pliki w komputerze – według nazw, wielkości (rosnąco lub malejąco) albo typów plików.

ZAPAMIĘTAJ!

Algorytmy sortowania są bardzo często stosowane w praktyce, korzysta z nich wiele aplikacji. Od strony informatycznej ocenia się je głównie na podstawie dwóch kryteriów – szybkości działania i pamięci potrzebnej na ich realizację. Wybór algorytmu sortowania zależy od wielu czynników, m.in. od rodzaju danych, od tego, czy są to dane losowe, prawie uporządkowane, z małego przedziału itp.

STRUKTURA DANYCH – LISTA

Dane można przechowywać w zmiennych. Jeśli jest ich więcej, warto skorzystać z listy – zmiennej, która pod swoją nazwą przechowuje wiele wartości. Dostęp do danych na liście uzyskuje się poprzez indeks każdego elementu.



Rys. 1. Elementy listy i ich indeksy

Poniżej zdefiniowano czteroelementową listę **t** i wypisano jej kolejne elementy.

```
1. t = [1, 12, -3, 8]
2.
3. print(t[0])          # 1
4. print(t[1])          # 12
5. print(t[2])          # -3
6. print(t[3])          # 8
```

Rys. 2. Definiowanie i wywoływanie elementów listy

ZAPAMIĘTAJ!

<nazwa listy> = [elementy listy oddzielone przecinkami]

ANALIZA LISTY

Podobnie jak w przypadku analizowania napisów, do przeglądania elementów listy warto wykorzystać pętlę **for**. Aby wypisać kolejne elementy listy, wystarczy zastosować pętlę dla danej sekwencji, tak jak to pokazano w poniższym przykładzie.

```
1. t = [1, 12, -3, 8]
2. for x in t:
3.     print(x)
```

Rys. 3. Wypisywanie kolejnych elementów listy **t**

Można też wykorzystać indeksowanie – wartość indeksu umieszczona po nazwie listy w nawiasie kwadratowym wskazuje odpowiedni element listy.

```
1. t = [1, 12, -3, 8]
2. for i in range(len(t)):
3.     print(t[i])
```

Rys. 4. Drugi sposób przeglądania kolejnych elementów listy

ZAPAMIĘTAJ!

Analizę elementów listy można przeprowadzić za pomocą pętli **for**:

- po elementach sekwencji

```
for <element> in <lista>:
    print(<element>)
```

- z wykorzystaniem indeksowania

```
for <indeks> in range(len(<lista>)):
    print(<element>[<indeks>])
```

Dodawanie elementów do listy można przeprowadzić na różne sposoby. W wyniku zastosowania metody **append()** nowe elementy zostaną wstawione na koniec listy. Jeśli korzysta się z operatora arytmetycznego **+**, nowe elementy zostaną wstawione na koniec lub początek listy w zależności od kolejności składników w zapisie dodawania. Aby wstawić element w wybranym miejscu, należy użyć dwuparametrowej metody **insert()**, gdzie pierwszy parametr jest indeksem, pod którym ma być wstawiony element podany jako drugi parametr. Przeanalizuj poniższy zapis i wypróbuj omówione sposoby dodawania elementów do listy.

```
1. t = [1, 12, -3, 8]
2. t.append(34)
3. print(t)
4. t = t + [-11]
5. print(t)
6. t = [23] + t
7. print(t)
8. t.insert(3, 5)
9. print(t)
```

Rys. 5. Różne sposoby dodawania kolejnych elementów do listy

W trakcie testowania sortowania ciągu liczb potrzebne są dane. Czasem łatwiej wygenerować je w postaci liczb losowych, niż wpisywać np. tysiąc liczb na klawiaturze. Do losowania liczb całkowitych służy funkcja **randint(a, b)** z modułu **random**. Daje ona w wyniku liczbę całkowitą **los** spełniającą nierówność $a \leq los \leq b$.

Ćwiczenie 1. Losowa lista liczb

Zdefiniuj funkcję **losuj(rozmiar, od, do)** służącą do generowania listy o podanej długości (**rozmiar**), której elementami są liczby losowe z podanego zakresu (**od** i **do**).

- Zaimportuj funkcję **randint**, a następnie zdefiniuj funkcję **losuj(rozmiar, od, do)**.
- Utwórz pustą listę.

- Wykorzystaj pętlę **for**, aby dodać kolejno wylosowane liczby do listy.

```
1. from random import randint
2.
3. def losuj(rozmiar, od, do):
4.     t = []          # lista pusta
5.     for i in range(rozmiar):
6.         t = t + [randint(od, do)]
7.     return t
```

- Sprawdź działanie funkcji dla kilku wartości, np.

```
1. t = losuj(16, 1, 10)
2. print(t)
```

Podczas sortowania często zachodzi potrzeba zamiany wartości dwóch zmiennych. Można to zapisać na dwa sposoby: przy użyciu zmiennej pomocniczej albo za pomocą specjalnej, właściwej dla Pythona konstrukcji do zamiany wartości zmiennych.

```
1. x = 5
2. y = 9
3.
4. pom = x
5. x = y
6. y = pom
7.
8. print(x)
9. print(y)
```

```
1. x = 5
2. y = 9
3.
4. x, y = y, x
5.
6. print(x)
7. print(y)
```

Rys. 6. Dwa sposoby zapisu zamiany wartości zmiennych

Ćwiczenie 2. Bąbelek

Porównaj kolejne pary elementów listy (pierwszy z drugim, drugi z trzecim itd.). Jeśli pierwszy z nich jest większy niż drugi, to zamień je miejscami. Przeanalizuj dane w tabeli (czy widzisz pewną prawidłowość w wynikowej liście?) i zdefiniowaną poniżej funkcję **babelek(t)**, której parametrem będzie lista liczb, a wynikiem - zmodyfikowana lista. Następnie znajdź i popraw błąd w skrypcie.

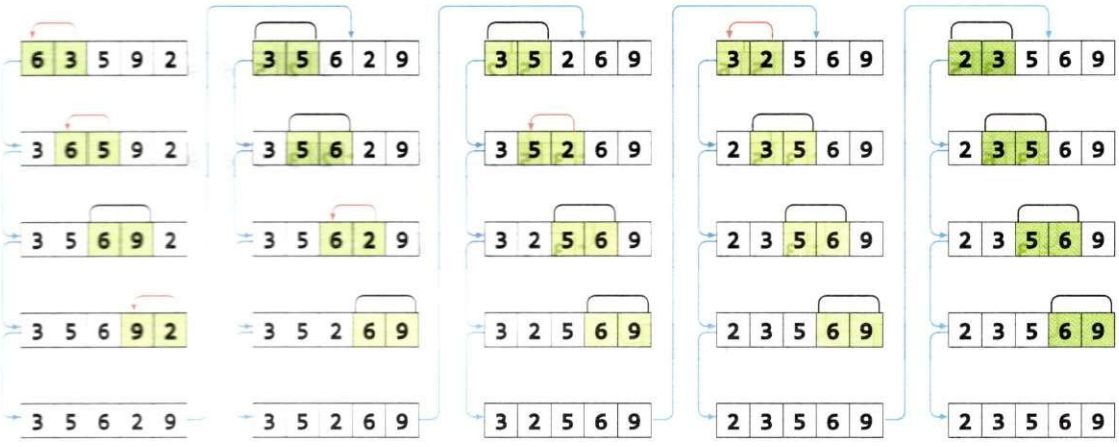
Wywołanie funkcji	Wynik
`babelek([16, 5, 12, 3, 12])`	`[5, 12, 3, 12, 16]`
`babelek([1, 16, 18, 3, 16, 9, 9])`	`[1, 16, 3, 16, 9, 9, 18]`

```
1. def babelek(t):
2.     for i in range(len(t) - 1):
3.         if t[i] < t[i + 1]:
4.             t[i], t[i + 1] = t[i + 1], t[i]
5.     return t
```

SORTOWANIE METODĄ BĄBELKOWĄ

Porządkowanie zbioru danych za pomocą metody bąbelkowej polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli elementy nie stoją względem siebie w takiej kolejności, w jakiej wykonywane jest porządkowanie. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

W jaki sposób posortować rosnąco ciąg składający się z liczb: 6, 3, 5, 9, 2? Dla listy o długości 5 należy porównać cztery razy dwa kolejne elementy listy i w razie konieczności je przestawić. Następnie należy zacząć porównanie od początku i powtarzać tak długo, dopóki nie będzie żadnych zmian.



Rys. 7. Sortowanie bąbelkowe liczb: 6, 3, 5, 9, 2. ⌒ – zmiana; ⌒ – bez zmian

Ćwiczenie 3. Sortowanie bąbelkowe

Przeanalizuj dane w tabeli i zdefiniuj funkcję **sort_b(t)**, której wynikiem będzie podana jako parametr lista **t**, posortowana metodą bąbelkową w porządku rosnącym.

Wywołanie funkcji	Wynik
`sort_b([6, 3, 15, 9, 2])`	`[2, 3, 6, 9, 15]`
`sort_b([11, 16, 8, 33, 6, 8, 1])`	`[1, 6, 8, 8, 11, 16, 33]`

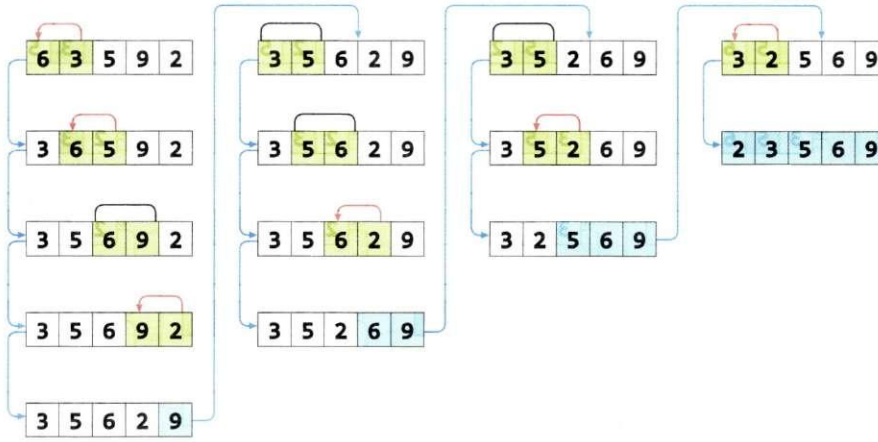
- Skorzystaj z rozwiązania poprzedniego ćwiczenia i rozszerz je o wielokrotne przeglądanie listy.

```
1. def sort_b(t):
2.     for j in range(len(t)):
3.         babelek(t)
4.     return t
```

- Sprawdź działanie skryptu z różnymi danymi. Możesz wykorzystać generowanie losowych wartości listy będącej parametrem.

```
1. print(sort_b([6, 3, 15, 9, 2]))
2. print(sort_b([11, 16, 8, 33, 6, 8, 1]))
3. t = losuj(16, 1, 10)
4. print(t)
5. print(sort_b(t))
```

Liczbę operacji tego algorytmu można zminimalizować. Przy pierwszym porównywaniu par liczb listy największa liczba powędruje na właściwe miejsce na liście, tak jak to było w funkcji w ćwiczeniu 2. Przy drugim przebiegu pętli przedostatnia liczba stanie na swoim miejscu itd.

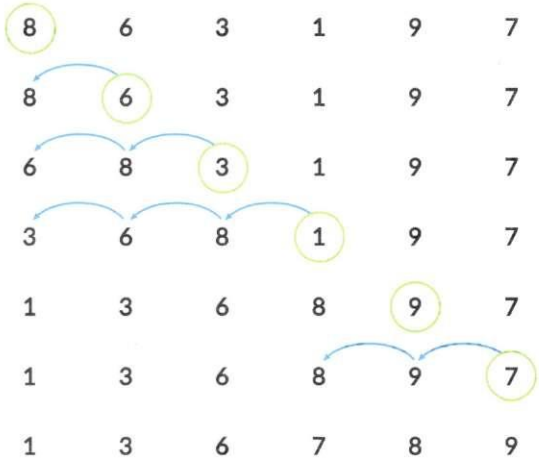


Rys. 8. Uproszczony algorytm sortowania bąbelkowego. ⌒ - zmiana; ⌒ - bez zmian

SORTOWANIE PRZEZ WSTAWIANIE

Algorytm sortowania przez wstawianie można porównać do sposobu układania kart w ręku przez gracza. Na początku bierze on pierwszą kartę, potem kolejno podnosi karty ze stołu. Każdą z nich porównuje z kartami, które trzyma już w ręce, i zastanawia się, gdzie ma ją wstawić. Gdy znajdzie miejsce, rozsuwa karty i wstawia wybraną kartę na przygotowane w ten sposób miejsce.

Przeanalizuj poniższy przykład sortowania ciągu liczb: 8, 6, 3, 1, 9, 7 oraz wizualizację zamieszczoną na stronie **https://pl.wikipedia.org/wiki/Sortowanie_przez_wstawianie**.



Rys. 9. Sortowanie przez wstawianie liczb: 8, 6, 3, 1, 9, 7

Ćwiczenie 4. Sortowanie przez wstawianie dla różnych danych

Obejrzyj animacje na stronie **www.toptal.com/developers/sorting-algorithms/insertion-sort** i porównaj działanie algorytmu sortowania przez wstawianie dla różnych danych (liczby losowe, prawie posortowane, posortowane w odwrotnej kolejności).

Ćwiczenie 5. Wstawianie elementów

Przeanalizuj dane w tabeli i zdefiniuj funkcję **`wstawianie(t, x)`**, której parametrami będą posortowana lista liczb oraz liczba **`x`**, a wynikiem – lista **`t`** z liczbą **`x`** wstawioną w odpowiednim miejscu.

Wywołanie funkcji	Wynik
`wstawianie([6, 13, 15, 19, 22], 18)`	`[6, 13, 15, 18, 19, 22]`
`wstawianie([1, 9, 12, 33, 46], 57)`	`[1, 9, 12, 33, 46, 57]`

- Wykorzystaj metodę **`insert()`** do wstawienia liczby podanej jako parametr w odpowiednie miejsce na liście.
- Rozpatrz trzy przypadki:
  - liczba jest mniejsza lub równa pierwszej liczbie z listy → wstawiasz daną liczbę przed pierwszą liczbą z listy;

- liczba jest większa lub równa ostatniej liczbie z listy → wstawiasz daną liczbę po ostatniej liczbie z listy;
- liczba jest większa od pierwszej liczby z listy i mniejsza od ostatniej liczby → przeglądasz listę i znajdujesz dwie kolejne liczby, między które dana liczba powinna zostać wstawiona.

- Zapisz całą funkcję.

```
1. def wstawianie(t, x):
2.     if x <= t[0]:
3.         t.insert(0, x)
4.         return t
5.     if x >= t[len(t) - 1]:
6.         t.insert(len(t), x)
7.         return t
8.     for i in range(len(t) - 1):
9.         if x >= t[i] and x < t[i + 1]:
10.            t.insert(i + 1, x)
11.            return t
```

Ćwiczenie 6. Sortowanie przez wstawianie

Przeanalizuj dane w tabeli i uzupełnij definicję funkcji **sort_w(t)**, której wynikiem będzie podana jako parametr lista **t**, posortowana metodą przez wstawianie w porządku rosnącym.

Wywołanie funkcji	Wynik
`sort_w([6, 3, 15, 9, 2])`	`[2, 3, 6, 9, 15]`
`sort_w([11, 16, 8, 33, 6, 8, 1])`	`[1, 6, 8, 8, 11, 16, 33]`

```
1. def sort_w(t):
2.     pom = []
3.     pom = pom + [t[0]]
4.     for i in range(1, len(t)):
5.         wstawianie(pom, #uzupełnij funkcję)
6.     return pom
```

- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 7. Sortowanie różnymi metodami

Obejrzyj animacje na stronie **www.toptal.com/developers/sorting-algorithms/random-initial-order** i porównaj sortowanie danych losowych wykonywane różnymi metodami. Zwróć uwagę na porównanie szybkości sortowania bąbelkowego i przez wstawianie.

PODSUMOWANIE

- Sortowanie jest bardzo często wykorzystywane w praktyce. Zanim posortuje się zbiór danych, trzeba określić klucz, według którego ma odbyć się sortowanie, i porządek, np. sortowanie rosnąco, malejąco.
- Sortowanie jest ważnym problemem informatycznym. Istnieje wiele algorytmów tego procesu – różnią się czasem działania i ilością pamięci potrzebnej na zapamiętanie tymczasowych danych.
- W sortowaniu bąbelkowym porównywane są dwa kolejne elementy. Zamienia się je miejscami, gdy stoją w kolejności innej niż oczekiwana. Przeglądanie elementów listy powtarza się do momentu, gdy przestawianie jest już niepotrzebne.
- Algorytm sortowania przez wstawianie polega na wstawianiu kolejnego elementu na właściwe miejsce. Należy zacząć od jednego elementu – oczywiście jest on posortowany. Drugi element wstawia się przed nim lub za nim, w zależności od oczekiwanego porządku sortowania. Tak samo postępuje się z następnymi elementami.

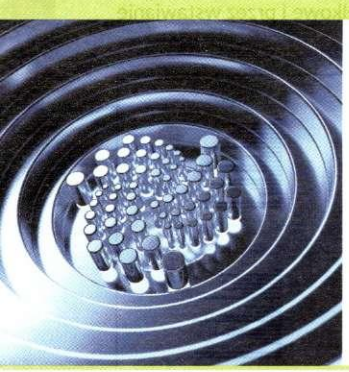
PYTANIA SPRAWDZAJĄCE

1. W jaki sposób można przeglądać wartości na liście?
2. Która z liczb – najmniejsza czy największa – znajdzie się na swoim miejscu po pierwszym przebiegu pętli w sortowaniu metodą bąbelkową w porządku malejącym?
3. Która z metod sortowania jest szybsza dla danych losowych?

ZADANIE DODATKOWE

Zadanie 1. Sortowanie bąbelkowe – wersja uproszczona

Zdefiniuj funkcję **`sort_b2(t)`**, której wynikiem będzie podana jako parametr lista **`t`**, posortowana uproszczoną metodą sortowania bąbelkowego.



5. Algorytmy zachłanne

- Dzielenie problemu na podproblemy
- Wydawanie reszty metodą zachłanną
- Podejście zachłanne kontra dynamiczne

ZADANIE NA START

Wydawanie reszty

Projektanci urządzeń do sprzedaży samoobsługowej produktów (np. gorących napojów, przekąsek) i usług (np. parkometry, biletomaty), którzy mają wyposażyć automat w system płatności za pomocą monet, muszą rozwiązać problem wydawania reszty: jak monetami o nominałach 20, 10, 5, 2, 1 wydać resztę w taki sposób, by użyć ich najmniej. Jakie będzie rozwiązanie dla reszty równej 67, a jakie dla 68? Co zmieni się po dołożeniu monety o wartości 4?

PROBLEM WYDAWANIA RESZTY I DZIAŁANIE ZACHŁANNE

Problem przedstawiony w zadaniu na start, polegający na wybraniu z danego zbioru monet o określonych nominałach takiej konfiguracji, by wydać żądaną kwotę przy użyciu minimalnej liczby monet, to zagadnienie z dziedziny algorytmiki, które nosi nazwę problemu wydawania reszty. Do dyspozycji są dwa rodzaje monet 1 i 3, a należy wydać resztę 4 oraz 7 na różne sposoby. Jak to zrobić? Punktem wyjścia do rozważań nad tym problemem jest podzielenie rozwiązania na etapy, by w każdym kroku wybrać najkorzystniejsze rozwiązanie częściowe.

Rys. 1. Problem wydawania reszty w przypadku dwóch rodzajów monet

Ćwiczenie 1. Trzy monety

Dysponujesz nieograniczoną liczbą monet o nominałach 5, 3 i 1 i masz wydać resztę przy użyciu minimalnej ich liczby. Przeanalizuj dane w tabeli i zdefiniuj funkcję **trzy(kwota)**, której wynikiem będzie minimalna liczba monet potrzebnych do wydania kwoty podanej jako parametr.

Wywołanie funkcji	Wynik
`trzy(11)`	3
`trzy(99)`	21

- Sformułuj algorytm. Ustaw wartość licznika wykorzystywanych monet na 0, a następnie rozpisz warunki dotyczące kolejnych nominałów (np. dopóki **kwota >= 5**, zmniejsz wartość zmiennej **kwota** o 5 i zwiększ wartość licznika o 1). Wynikiem działania całego algorytmu powinna być ostateczna wartość licznika.
- Zdefiniuj funkcję **trzy(kwota)**. Na potrzeby testowania usuń znak # tak, by wypisywały się wykorzystane monety.

```
1. def trzy(kwota):
2.     ile = 0
3.     while kwota >= 5:
4.         kwota = kwota - 5
5.         ile += 1
6.         #print("5")
7.     while kwota >= 3:
8.         kwota = kwota - 3
9.         ile += 1
10.        #print("3")
11.    while kwota >= 1:
12.        kwota = kwota - 1
13.        ile += 1
14.        #print("1")
15.    return ile
```

- Można funkcję zapisać krócej z zastosowaniem dzielenia całkowitego oraz reszty z dzielenia.

```
1. def trzy(kwota):
2.     ile = kwota // 5
3.     kwota = kwota % 5
4.     ile = ile + kwota // 3
5.     kwota = kwota % 3
6.     ile = ile + kwota
7.     return ile
```

- Sprawdź działanie skryptu z różnymi danymi – uwzględnij przypadki, w których powinny być wykorzystane tylko monety jednego rodzaju lub wszystkie (zarówno jednokrotnie, jak i wielokrotnie).

Czas rozważyć przypadek bardziej ogólny – jak zapisać algorytm wydawania reszty dla zbioru monet o nominałach uporządkowanych malejąco? (Należy założyć, że jest możliwe wydanie reszty o podanych nominałach).

1. Wczytaj kwotę do wydania reszty i zapamiętaj ją na zmiennej **`kwota`**.
2. Przypisz do zmiennej **`ile`** liczbę dotychczas wykorzystanych monet (tj. **`0`**).
3. Ze zbioru monet wybierz największą nierozpatrywaną monetę i przypisz ją do zmiennej **`moneta`**.
4. Zwiększ licznik wykorzystanych monet o **`kwota // moneta`**. Zmień kwotę **`reszta`** na **`reszta % moneta`**
5. Jeśli jest jakaś nierozpatrywana moneta, to przejdź do kroku 3.
6. Wypisz liczbę wykorzystanych monet (zmienna **`ile`**).

Powyższy algorytm charakteryzuje się tym, że w danym obrocie pętli bierze monetę o najwyższym nominale, czyli wybierana jest lokalnie najlepsza opcja.

ZAPAMIĘTAJ!

Algorytm zachłanny charakteryzuje się tym, że w celu wyznaczenia rozwiązania w każdym kroku dokonuje zachłannego, tj. najlepszego w danym momencie wyboru rozwiązania podproblemu. Innymi słowy, jest to decyzja lokalnie optymalna.

Ćwiczenie 2. Wiele monet

Dysponujesz nieograniczoną liczbą monet o nominałach 20, 10, 5, 2 i 1 i masz wydać resztę przy użyciu minimalnej ich liczby. Przeanalizuj dane w tabeli i zdefiniuj funkcję **`wiele(kwota)`**, której wynikiem będzie minimalna liczba monet potrzebnych do wydania kwoty podanej jako parametr.

Wywołanie funkcji	Wynik
`wiele(11)`	**2**
`wiele(99)`	**8**

- Wykorzystaj skrypt z ćwiczenia 1, tylko zamiast konkretnych wartości użyj listy monet o określonych nominałach, uporządkowanych od największej wartości do najmniejszej.
- Zdefiniuj funkcję **`wiele(kwota)`**.

```
1. def wiele(kwota):
2.     nominaly = [20, 10, 5, 2, 1]
3.     ile = 0
4.     for x in nominaly:
5.         #print(x, kwota // x)
6.         ile = ile + kwota // x
7.         kwota = kwota % x
8.     return ile
```

- Sprawdź działanie skryptu z różnymi danymi – uwzględnij wartości brzegowe, np. dobierz kwotę tak, aby były wypłacane tylko monety o nominale 20; każda moneta jeden raz; każda moneta maksymalnie trzy razy.

Ćwiczenie 3. Ograniczona liczba monet

Dysponujesz podaną liczbą monet o określonych nominałach i masz wydać resztę przy użyciu minimalnej ich liczby. Przeanalizuj dane w tabeli i zdefiniuj funkcję **monety(kwota, nominaly, sztuki)**, której wynikiem będzie minimalna liczba monet potrzebnych do wydania kwoty podanej jako parametr.

Wywołanie funkcji	Wynik
`monety(11, [20, 10, 5, 2, 1], [100, 100, 100, 100, 100])`	**2**
`monety(99, [20, 10, 5, 1], [1, 10, 1, 10])`	**13**

- Problem wymaga zmodyfikowania algorytmu z ćwiczenia 2. Podczas sprawdzania, czy można wydać resztę określonym nominałem, trzeba uwzględniać nie tylko wartość nominału, lecz także liczbę dostępnych monet. Dlatego w przypadku każdego nominału należy dodatkowo sprawdzić, czy są dostępne monety o tej wartości, a gdy zostaną wykorzystane – pomniejszyć licznik wykorzystanych monet.
- Skorzystaj ze skryptu w ćwiczeniu 2 i zdefiniuj funkcję **monety(kwota, nominaly, sztuki)**.
  - Zauważ, że liczba dostępnych monet **sztuki[i]** wskazywanych przez **nominaly[i]** jest tutaj kolejnym elementem listy liczonym od 0.

```
1. def monety(kwota, nominaly, sztuki):
2.     ile = 0
3.     for i in range(len(nominaly)):
4.         m = min(sztuki[i], kwota // nominaly[i])
5.         if m != 0:
6.             ile = ile + m
7.             sztuki[i] = sztuki[i] - m
8.             kwota = kwota - m * nominaly[i]
9.             #print(nominaly[i], m)
10.    return ile
```

- Sprawdź działanie skryptu z różnymi danymi – uwzględnij przypadki, w których liczba poszczególnych monet jest duża, by uzyskać wyniki identyczne z tymi w poprzednim ćwiczeniu.
- Czy zmniejszenie liczby monet wpływa na rozwiązanie?

POPRAWNOŚĆ ALGORYTMU ZACHŁANNEGO

Czy zapisany w ćwiczeniu 3 algorytm jest poprawny – czy faktycznie gwarantuje wydanie reszty najmniejszą liczbą nominałów?

Odpowiedź zależy od zbioru rozpatrywanych nominałów. O ile trudno udowodnić, dla jakich nominałów algorytm daje poprawną odpowiedź, o tyle łatwiej wskazać kontrprzykład, w którym algorytm zachłanny nie daje optymalnego rozwiązania.

Rozważ zbiór nominałów 5, 4, 1 i kwotę 8. Jeśli zastosujesz metodę zachłanną, to otrzymasz cztery monety: 5, 1, 1, 1. Istnieje jednak lepsze rozwiązanie: 4, 4. Czy umiesz znaleźć inne przykłady, dla których algorytm zachłanny nie daje poprawnych wyników?

Mimo że dla wielu zbiorów nominałów algorytm daje poprawny wynik (np. w przypadku systemu monetarnego obowiązującego w Polsce), nie można stwierdzić, że sprawdza się on w każdym przypadku. Wtedy trzeba szukać innego rozwiązania.

ZAPAMIĘTAJ!

Metody zachłanne nie gwarantują otrzymania optymalnego rozwiązania wszystkich problemów, do których mogą być wykorzystane – istnieją jednak problemy, dla których te metody zawsze zwracają rozwiązanie optymalne.

PROGRAMOWANIE DYNAMICZNE

Do rozwiązania rozpatrywanego problemu można wykorzystać programowanie dynamiczne. Problem jest wówczas dzielony na podproblemy, które są od siebie zależne (decyzja podejmowana na danym etapie zależy od decyzji podjętych na poprzednich etapach). W ten sposób pewnych rzeczy nie trzeba obliczać wielokrotnie – wystarczy korzystać z dotychczasowych rozwiązań pośrednich, a następnie wyszukiwać kolejne rozwiązania.

POSZUKAJ W INTERNECIE

Sprawdź, do czego odnosi się słowo „programowanie" w nazwie „programowanie dynamiczne".

Załóżmy, że dysponujesz monetami o nominałach 1, 4 i 5 i masz wydać kwotę 10. Przeanalizuj tabelę na stronie obok. W pierwszej kolumnie wpisano kwoty, w drugiej kolumnie – liczbę monet potrzebną do uzbierania kwoty za pomocą nominału 1, w trzeciej – za pomocą nominałów 1 i 4, w czwartej – za pomocą nominałów 1, 4 i 5.

	Monety do dyspozycji		
Kwota	1	1, 4	1, 4, 5
1	1	1	1
2	2	2	(2)
3	3	3	(3)
4	4	1	(1)
5	5	2	1
6	6	3	2
7	7	(4)	3
8	8	(2)	2
9	9	(3)	2
10	10	4	2

Rozwiązanie z użyciem monet o nominałach 1, 4 i 5 wymaga 3 monet. Jest to minimum z 4 (rozwiązanie dla 1 i 4) oraz 2 + 1 (rozwiązanie dla 1, 4 i 5 dla kwoty 7 – 5 = 2).

Rozwiązanie z użyciem monet o nominałach 1, 4 i 5 wymaga 2 monet. Jest to minimum z 2 (rozwiązanie dla 1 i 4) oraz 3 + 1 (rozwiązanie dla 1, 4 i 5 dla kwoty 3).

Rozwiązanie z użyciem monet o nominałach 1, 4 i 5 wymaga 2 monet. Jest to minimum z 3 (rozwiązanie dla 1 i 4) oraz 1 + 1 (rozwiązanie dla 1, 4 i 5 dla kwoty 4).

Za każdym razem jako rozwiązanie wybiera się minimalną liczbę z dwóch rozwiązań, dlatego dla kwoty równej 8 wskazano rozwiązanie dla dwóch czwórek, a nie piątki i trzech jedynek.

Ogólnie w każdym kolejnym kroku należy wziąć minimum z dwóch liczb: liczby monet potrzebnych do odliczenia kwoty bez największej monety oraz z jej wykorzystaniem. Przy czym w drugim przypadku trzeba znaleźć wcześniej wyliczoną wartość w tabeli dla pozycji wartość – nominał.

Ćwiczenie 4. Metoda zachłanna i dynamiczna

Dysponujesz nieograniczoną liczbą monet o nominałach 1, 2, 5 i 10 i masz wydać resztę przy użyciu minimalnej ich liczby. Przedstaw pisemnie rozwiązanie dla kwoty 18. Zastosuj podejście zachłanne i dynamiczne. Wykorzystaj tabelę zamieszczoną powyżej.

PODSUMOWANIE

- Algorytm zachłanny charakteryzuje się tym, że w celu wyznaczenia rozwiązania przy każdym kroku dokonuje najlepszego w danym momencie – zachłannego – wyboru rozwiązania podproblemu. Algorytm nie jest uniwersalny, daje poprawne wyniki tylko dla pewnej grupy problemów.
- W podejściu dynamicznym dzieli się problem na podproblemy, ale decyzje dla podproblemów podejmuje się z wykorzystaniem danych uzyskanych wcześniej. Zależność można opisać rekurencyjnie.

PYTANIA SPRAWDZAJĄCE

1. Czym charakteryzuje się podejście zachłanne przy rozwiązywaniu zadań? Wskaż najważniejszą cechę charakterystyczną.
2. Jaki problem poza wydawaniem reszty można rozwiązań metodą zachłanną?
3. Czym charakteryzuje się podejście dynamiczne?

ZADANIA DODATKOWE

Zadanie 1. Jak spakować plecak

Przeanalizuj dane w tabeli i zdecyduj, jak spakować plecak w taki sposób, żeby zabrać najwartościowsze rzeczy, ale nie więcej niż 40 kg.

	Masa (kg)	Cena (zł)
1	35	45
2	15	40
3	20	35
4	10	25
5	20	25
6	10	5

Zadanie 2. Nowe zestawy monet

Dobierz tak zestaw monet, aby podejście zachłanne zastosowane w ćwiczeniu 4 nie dawało optymalnego rozwiązania. Przedstaw pisemnie rozwiązanie dla metody zachłannej i dynamicznej.