ALGORYTMIKA I PROGRAMOWANIE W PYTHONIE

- Składnia języka, operatory, zmienne
- Funkcje, instrukcje warunkowe
- Algorytm naiwny wyszukiwania wzorca w tekście
- Porównywanie, przeszukiwanie, przetwarzanie napisów
- Szyfry przestawieniowe, szyfr Cezara

⌜·O>L·⌜>, CZYLI SZYFRY

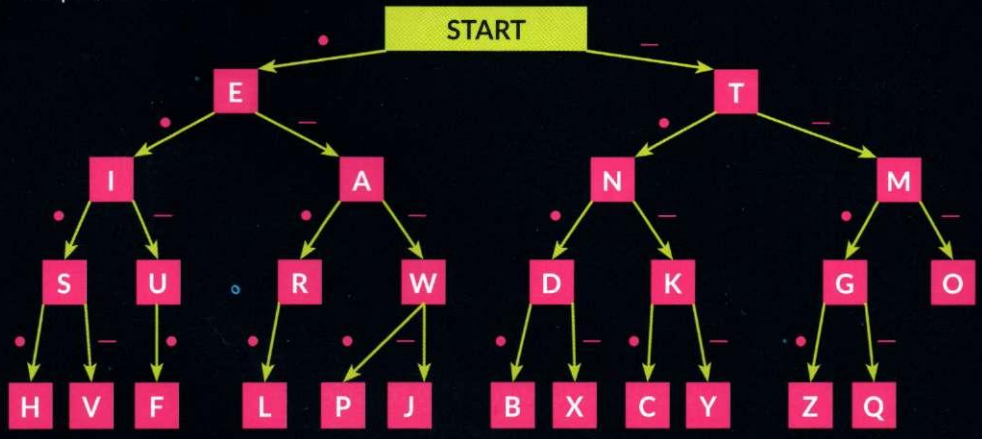
ALFABET MORSE'A

Litery, liczby i znaki specjalne w alfabecie opracowanym w 1832 r. przez amerykańskiego malarza Samuela Morse'a są przedstawiane jako kropki i kreski. Kod, który obecnie używany jest głównie przez radioamatorów, może służyć do szyfrowania tekstu pisanego oraz sygnalizacji światłem, dźwiękiem czy chorągiewkami.

PRZYKŁAD: SOS → /• • • /– – – /• • •/

LITERA	KOD	LITERA	KOD
A	• –	N	– •
B	– • • •	O	– – –
C	– • – •	P	• – – •
D	– • •	Q	– – • –
E	•	R	• – •
F	• • – •	S	• • •
G	– – •	T	–
H	• • • •	U	• • –
I	• •	V	• • • –
J	• – – –	W	• – –
K	– • –	X	– • • –
L	• – • •	Y	– • – –
M	– –	Z	– – • •

W odczytaniu wiadomości zapisanej za pomocą alfabetu Morse'a może pomóc drzewo.



Szyfrowanie polega na przekształcaniu wiadomości, tzw. tekstu jawnego, w szyfrogram tak, by uniemożliwić osobom niepowołanym odczytanie informacji.

ROT13

Alfabet o długości 26 znaków dzieli się na dwie części, a następnie każdej literze z pierwszej połowy przypisuje się odpowiadającą jej literę z drugiej połowy i na odwrót, np.: A → N, N → A, J → W, W → J. Ten sam algorytm wykorzystywany jest do szyfrowania i do deszyfrowania wiadomości. Jest to szczególny przypadek szyfru Cezara z kluczem 13.

PRZYKŁAD:
TAJNE HASLO BRZMI – CUKIEREK. → GNWAR UNFYB OEMZV – PHXVRERX.

A	B	C	D	E	F	G	H	I	J	K	L	M
↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓↑	↓↑
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

PŁOTEK

Jest to szyfr przestawieniowy, w którym klucz liczbowy oznacza „wysokość" tworzonych schodków. Kolejne litery tekstu jawnego zapisywane są w dół, a potem w górę. Zaszyfrowany tekst powstaje poprzez połączenie kolejnych liter pierwszego wiersza, drugiego, itd.

PRZYKŁAD:
KASZTANY → KTAZAYSN

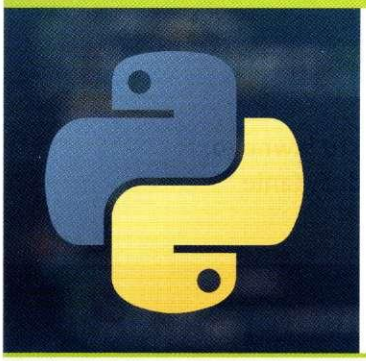
K				T			
	A		Z		A		Y
		S				N	

CZEKOLADKA

W tym szyfrze rysunek podpowiada graficzny zapis liter. Chcąc zaszyfrować literę, należy narysować „kostkę czekoladki", stawiając kropkę bliżej tej krawędzi, przy której jest szyfrowana litera. Aby zapisać litery T, U, Y lub W, rysuje się odpowiedni „dzióbek", do zapisania litery Z służy okrąg.

PRZYKŁAD:
NAMIOT →

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	R	S



14. Podstawy pracy w środowisku Python

- Wprowadzenie do programowania w języku Python
- Praca w trybie bezpośrednim i w pliku
- Definiowanie prostych funkcji

ZADANIE NA START

Programowanie w Pythonie

Python to język interpretowany, którego używają zarówno początkujący programiści, jak i profesjonaliści. Doskonale sprawdza się jako narzędzie do tworzenia bardziej lub mniej złożonych aplikacji desktopowych, pracujących pod kontrolą systemów operacyjnych Windows, Linux czy OS, dynamicznych stron internetowych oraz zaawansowanych aplikacji webowych. Na przykład Google wykorzystuje Pythona w wyszukiwarce, amerykańska National Security Agency – w kryptografii oraz analizach wywiadowczych, firmy Intel, Hewlett-Packard czy IBM – w testowaniu urządzeń. Znajdź trzy aplikacje napisane w języku Python.

ABC JĘZYKA PYTHON

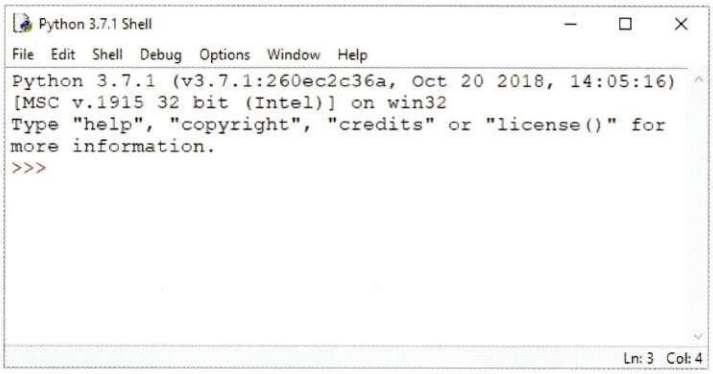
W systemie Windows pracę w środowisku Python należy rozpocząć od zainstalowania tego środowiska na komputerze. W Ubuntu Linuksie język ten stanowi standardowy element systemu. Środowisko Python w wersji 3.* odpowiedniej dla danego systemu należy pobrać z serwisu **https://python.org**. Oprócz rekomendowanej opcji instalacji programu dla wszystkich użytkowników warto zaznaczyć opcję umożliwiającą uruchamianie skryptów w wierszu polecenia z dowolnej lokalizacji.

POSZUKAJ W INTERNECIE

Poszukaj w internecie wywiadów z twórcą Pythona Guidem van Rossumem, aby dowiedzieć się więcej o nim i jego filozofii oraz o pochodzeniu nazwy języka.

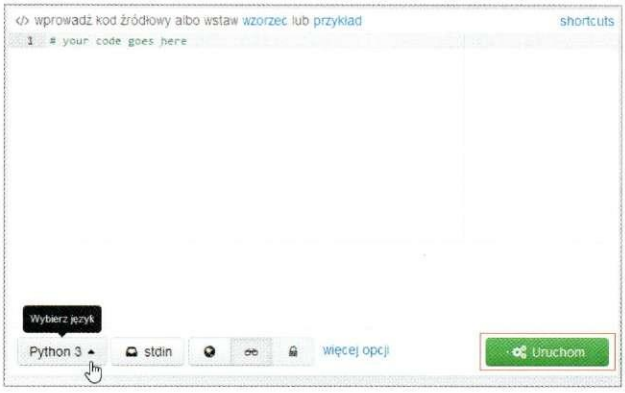
Skrypty Pythona można zapisywać w dowolnym edytorze tekstu, ale wygodniej jest używać zintegrowanego środowiska deweloperskiego (IDE), zawierającego różne funkcje, np. kolorowania składni. Osobom początkującym wystarczy **IDLE** – prosty edytor stanowiący część domyślnego pakietu instalacyjnego.

Po uruchomieniu IDLE otworzy się okno **Shell** – powłoka systemowa działająca jako interpreter, który wyświetla wynik działania skryptu zapisanego w nim w trybie bezpośrednim albo programu utworzonego wcześniej w edytorze. Trzy nawiasy ostrokątne, tzw. znak zachęty, oznaczają, że interpreter czeka na wprowadzenie polecenia.



Rys. 1. Okno Shell edytora IDLE

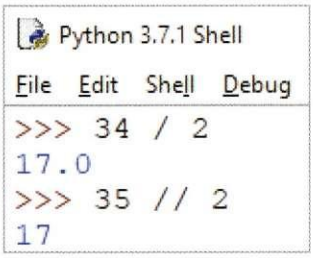
W ostatnich latach coraz więcej osób pracuje w chmurze. Programowanie online umożliwia m.in. serwis **ideone.com** stworzony przez polską firmę z Trójmiasta. Serwis ten obsługuje ponad 40 języków programowania. Warto założyć w nim konto i ustawić domyślny język programowania, język interfejsu oraz włączyć podświetlanie składni, a także używać notatek do opisu programu oraz etykiet do organizacji swoich kodów. Wszystkie skrypty programów zapisywane są w chmurze, ale można je pobrać na dysk i skorzystać z nich ponownie.



Rys. 2. W serwisie **ideone.com** kursor wskazuje wybór języka, a czerwona ramka – uruchomienie programu

OPERATORY

Python obsługuje operatory arytmetyczne i porównania. Do siedmiu podstawowych **operatorów arytmetycznych** należą: dodawanie (+), odejmowanie (–), mnożenie (*), potęgowanie (**), dzielenie rzeczywiste (/), dzielenie całkowite (//) oraz reszta z dzielenia całkowitego (%). W przypadku dzielenia rzeczywistego iloraz jest liczbą rzeczywistą, dlatego wynikiem dzielenia 4 przez 2 jest dla Pythona liczba 2.0 (część całkowitą w tym języku zawsze oddziela od części dziesiętnej kropka). W przypadku dzielenia całkowitego wynikiem może być tylko liczba całkowita – iloraz całkowity dwóch liczb bez reszty.



Rys. 3. Dzielenie rzeczywiste i całkowite w trybie bezpośrednim IDLE

Zastanów się, jak za pomocą potęgowania obliczyć pierwiastek kwadratowy z liczby?

Ćwiczenie 1. Kupowanie obcej waluty

Oblicz, ile musisz zapłacić w złotówkach za 126 euro (aktualny kurs wymiany sprawdź na stronie NBP).

- Zapisz odpowiedni wzór: przemnóż daną kwotę przez aktualny kurs wymiany (pamiętaj o zastąpieniu przecinka kropką) – np. dla 1 euro równego 4,20 zł kod przyjmuje poniższą postać:

```
126 * 4.2
```

Operatory porównania wykorzystywane w języku Python zaprezentowano w poniższej tabeli wraz z przykładami.

Operatory porównania			
==	równe	3 == 3 → True	4 == 3 → False
!=	różne	3 != 2 → True	3 != 3 → False
<	mniejsze	3 < 4 → True	4 < 3 → False
<=	mniejsze równe	4 <= 4 → True	4 <= 3 → False
>	większe	4 > 3 → True	3 > 4 → False
>=	większe równe	4 >= 4 → True	4 >= 5 → False

Tab. 1. Operatory porównania i przykłady

Wynikami porównań są wartości logiczne **prawda** lub **fałsz**. Należy pamiętać, że w porównaniu znak równości w Pythonie jest podwójny. Pojedynczy znak = to operator przypisania, za pomocą którego przypisuje się zmiennej pewną wartość.

Ćwiczenie 2. Porównanie cen wyrażonych w różnych walutach

Porównaj ceny tych samych butów, które w Grecji kosztują 49 euro, a w Polsce 200 zł (aktualny kurs wymiany sprawdź na stronie NBP).

- **Sposób 1** – przelicz cenę butów w Grecji na złotówki, a potem porównaj obie wartości.
- **Sposób 2** – zapisz oba działania w jednym wierszu z wykorzystaniem operatorów arytmetycznych i porównania. Gdy 1 euro = 4,20 zł, kod w IDLE może przyjąć poniższą postać:

```
>>> 49 * 4.2 >= 200
True
```

ZMIENNE

Zmienne przechowują wartości określonego typu, m.in. liczby całkowite, liczby rzeczywiste, wartości logiczne i łańcuchy znaków. Każda zmienna w Pythonie powinna mieć krótką i znaczącą nazwę (bez polskich liter i spacji), np. **n** albo **liczba**. Należy przy tym pamiętać, że **wielkość liter jest istotna** – **N** i **n** to różne zmienne. Aby nadać zmiennej wartość, należy zastosować operator przypisania **=**. W każdej chwili można z tej wartości skorzystać lub ją modyfikować.

```
>>> x = 34
>>> y = 66
>>> x + y
100
```

Rys. 4. Obliczanie sumy zmiennych całkowitych **x** oraz **y**

Ćwiczenie 3. Średni dystans

W czasie wakacji uczniowie na obozie wędrownym przeszli w ciągu kolejnych trzech dni 12, 14 i 18 km. Zdefiniuj odpowiednie zmienne i oblicz średnią długość pokonywanych tras.

- Zdefiniuj zmienne i przypisz im podane wartości.

```
a = 12
b = 14
c = 18
```

- Zapisz odpowiedni wzór, za pomocą którego wyznaczysz średnią podanych wartości.

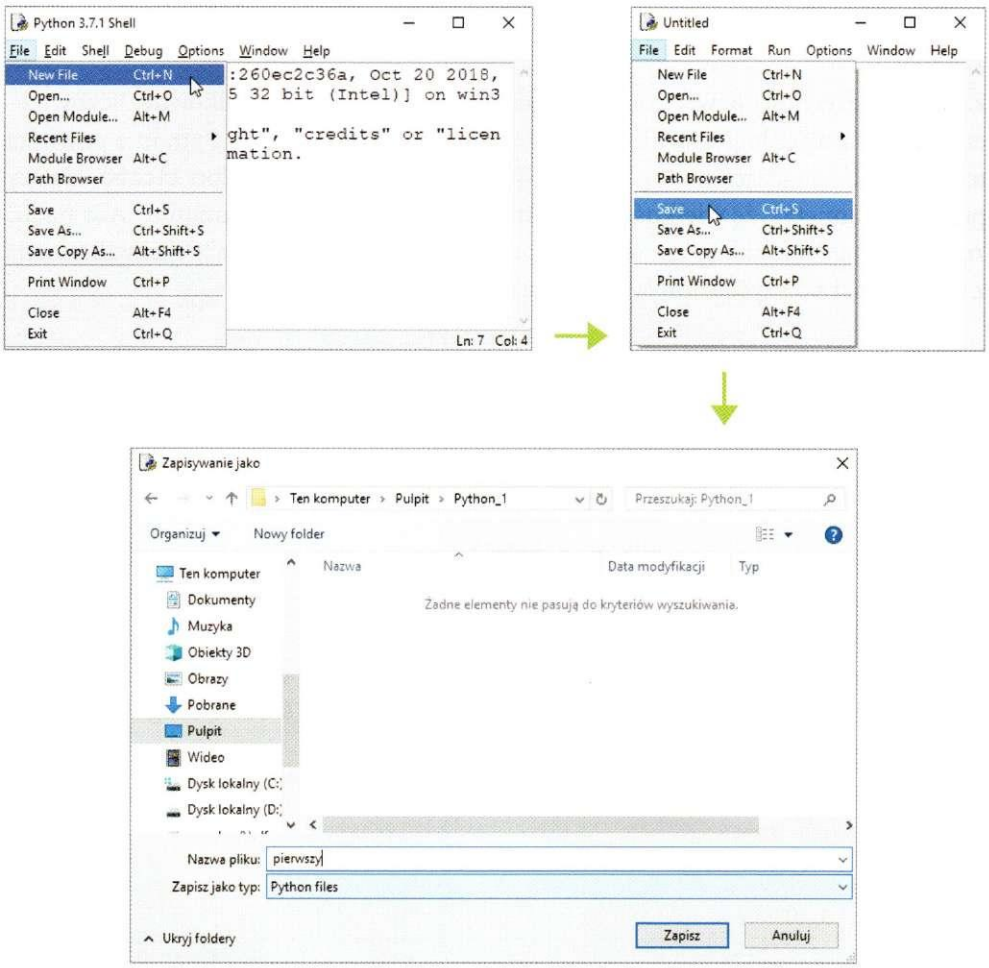
```
(a + b + c)/3
```

- Sprawdź odpowiedź.

PRACA W EDYTORZE

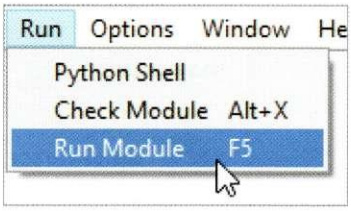
Praca w edytorze kodu polega na pisaniu poleceń zgodnie z regułami obowiązującymi dla danego języka. Te polecenia w dalszej kolejności są analizowane przez interpreter, a następnie wykonywane przez komputer.

Aby rozpocząć pracę w edytorze IDLE, należy kliknąć **File** na pasku menu i wybrać opcję **New File**. Otworzy się niezatytułowane okno – edytor. Następnie w oknie edytora należy wybrać **File** i opcję **Save** lub **Save as**, nazwać plik i zapisać go w preferowanej lokalizacji. W ten sposób powstanie plik z rozszerzeniem PY.



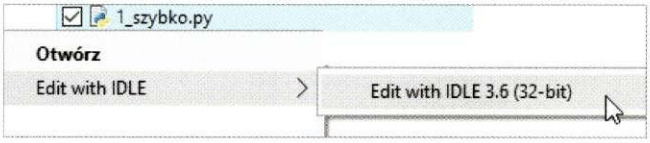
Rys. 5. Tworzenie pliku o nazwie **pierwszy** w IDLE

Aby uruchomić program, należy wybrać na pasku menu **Run → Run Module**. Gdy pojawi się okno z pytaniem, czy zapisać zmiany, trzeba kliknąć **OK**. Warto pamiętać, że każde uruchomienie pliku Pythona powoduje jego zapisanie.



Rys. 6. Uruchamianie programu w IDLE

Aby ponownie edytować utworzony plik, trzeba go zaznaczyć, a następnie z menu podręcznego wybrać polecenie **Edit with IDLE**.



Rys. 7. Otwieranie pliku PY do edycji w IDLE

Aby rozpocząć pracę w edytorze w serwisie **ideone.com**, wystarczy w przeglądarce wpisać adres serwisu i po wgraniu strony można zacząć pracę. Aby uruchomić program, należy wybrać opcję **Uruchom**. Wszystkie zapisane programy znajdują się w sekcji **Moje kody**. Każdy z nich można ponownie otworzyć, duplikować lub pobrać na dysk.

WIEM WIĘCEJ

Podczas pisania programu należy dbać o odpowiednie standardy kodowania. Z regułami i zaleceniami dotyczącymi formatowania kodu w Pythonie (*Style Guide for Python Code*) można się zapoznać na stronie **www.python.org/dev/peps/pep-0008**.

POLECENIE PRINT

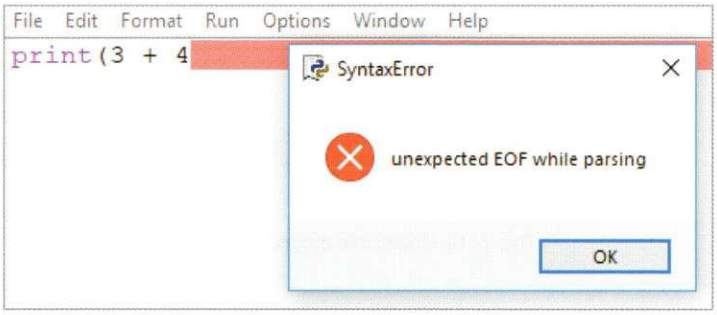
Instrukcja **print** służy do pisania znaków. Możesz wpisać tekst (ciąg znaków), liczbę albo zmienną.

ZAPAMIĘTAJ!

```
print("<ciąg znaków>")
print(<liczba>)
print(<zmienna>)
```

Aby dodać pusty wiersz, można skorzystać z polecenia **print("")**.

Jeśli do zapisu wkradnie się błąd składni, pojawi się komunikat **SyntaxError** – kolorowy pasek w IDLE lub szczegółowa informacja o kompilacji w serwisie **ideone.com** wskaże, który wiersz należy skorygować.



edytuj klonuj pobierz

```
1. print (3 + 4
```

Błąd kompilacji #stdin błąd kompilacji #stdout 0s 0KB

stdin

```
Standard input is empty
```

informacja o kompilacji

```
Traceback (most recent call last):
  File "/usr/lib/python3.5/py_compile.py", line 125, in compile
    _optimize=optimize)
  File "<frozen importlib._bootstrap_external>", line 735, in source_to_code
  File "<frozen importlib._bootstrap>", line 222, in _call_with_frames_removed
  File "./prog.py", line 1
    print (3 + 4
               ^
SyntaxError: unexpected EOF while parsing

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.5/py_compile.py", line 129, in compile
    raise py_exc
py_compile.PyCompileError:   File "./prog.py", line 1
    print (3 + 4
               ^
SyntaxError: unexpected EOF while parsing
```

stdout

```
Standard output is empty
```

Rys. 8. Przykładowy komunikat informujący o braku nawiasu zamykającego w IDLE i w serwisie **ideone.com**

DEFINIOWANIE FUNKCJI

Funkcje pozwalają podzielić większy problem na mniejsze, względnie niezależne części, a także przyśpieszają i ułatwiają pisanie programu, który staje się dzięki nim bardziej zwięzły i przejrzysty. Definicję funkcji zaczyna się od słowa kluczowego **def**, następnie podaje się nazwę funkcji (bez polskich znaków, spacji) i wstawia nawiasy, w których można umieścić parametr lub parametry lub nie umieszczać niczego (funkcja bez parametru). Deklarację należy zakończyć dwukropkiem, który zagwarantuje, że po naciśnięciu klawisza **Enter** w następnym wierszu powstanie wcięcie. Kolejne wiersze kodu zawierają instrukcje do wykonania. Polecenie **return**, które przekazuje szukaną wartość, kończy działanie funkcji.

ZAPAMIĘTAJ!

```
def <nazwa funkcji>(<parametr>):
    <instrukcje>
```

Ćwiczenie 4. Przeliczanie prędkości

Samolot z Warszawy do Rzymu pokonuje dystans 1320 km. Przeanalizuj dane w tabeli i zdefiniuj funkcję **jak_szybko(t)**, której parametrem jest czas przelotu samolotu w godzinach, a wynikiem średnia prędkość tego samolotu.

Wywołanie funkcji	Wynik
jak_szybko(2.5)	**528.0**
jak_szybko(3)	**440.0**

- Aby wyznaczyć średnią prędkość, należy podzielić drogę przez czas.
- Zdefiniuj funkcję **jak_szybko(t)**, np.

```
1. def jak_szybko(t):
2.     s = 1320
3.     return s / t
```

- Sprawdź działanie skryptu z różnymi danymi.
  - W IDLE wybierz z menu **Run** → **Run Module**, kliknij **OK**, a następnie w oknie Shell napisz wywołanie funkcji z odpowiednim parametrem w nawiasie.

```
>>> jak_szybko(2.5)
528.0
>>> jak_szybko(3)
440.0
```

- W serwisie **ideone.com** pod definicją funkcji dodaj polecenie **print** i umieść wywołanie funkcji z odpowiednim parametrem w nawiasie.

ZAPAMIĘTAJ!

W Pythonie polecenia grupowane są w bloki, które uzyskuje się przez **indentację**, tj. wcięcia w wierszach. Jeśli kilka wierszy stanowi jeden blok, to wszystkie te wiersze muszą być poprzedzone jednakowymi wcięciami.

Ćwiczenie 5. Przeliczanie jednostek

Mila lądowa, jednostka długości stosowana w krajach anglosaskich, równa jest 1,609 km. Przeanalizuj dane w tabeli i zdefiniuj funkcję **zamiana(m)**, której parametrem jest odległość w milach lądowych, a wynikiem ta sama odległość w kilometrach.

Wywołanie funkcji	Wynik
zamiana(10)	**16.09**
zamiana(12)	**19.308**

- Aby zamienić mile na kilometry, należy pomnożyć odległość w milach przez 1,609.
- Zdefiniuj funkcję **zamiana(m)**, np.

```
1. def zamiana(m):
2.     return m * 1.609
```

- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 6. O której godzinie pociąg dotrze do celu?

Nocny pociąg z Krakowa do Paryża wyjeżdża ze stacji o godzinie 21. Przeanalizuj dane w tabeli i zdefiniuj funkcję **godzina(n)**, której parametrem jest czas przejazdu pociągu w godzinach, a wynikiem godzina dojazdu do celu.

Wywołanie funkcji	Wynik
godzina(14)	**11**
godzina(20)	**17**

- 21 + 14 = 35, 21 + 20 = 41. Doba trwa 24 godziny, dlatego aby obliczyć czas przyjazdu pociągu, należy wyznaczyć resztę z dzielenia wyniku dodawania przez 24.
- Zdefiniuj funkcję **godzina(n)** – wykorzystaj do obliczeń resztę z dzielenia całkowitego.

```
1. def godzina(n):
2.     return (21 + n) % 24
```

- Sprawdź działanie skryptu z różnymi danymi.

ZAPAMIĘTAJ!

Ważną umiejętnością programistyczną jest **zapis w języku formalnym** tego, co potrafimy wyliczyć bez komputera. Podczas rozwiązywania zadań staramy się najpierw zrozumieć problem, przeanalizować go na kilku przykładach, potem zaproponować algorytm i zapisać go w języku programowania. Nie wolno zapomnieć o testowaniu. Wypracowanie takiej metody postępowania ułatwia rozwiązanie różnych problemów, nie tylko tych wywodzących się z informatyki.

Ćwiczenie 7. Która godzina będzie w Nowym Jorku?

Sprawdź, jaka strefa czasowa obowiązuje w Nowym Jorku, a następnie przeanalizuj dane w tabeli i zdefiniuj funkcję **godz_nj(g)**, której parametrem jest pełna godzina w Warszawie, a wynikiem aktualna pełna godzina w Nowym Jorku.

Wywołanie funkcji	Wynik
godz_nj(18)	**12**
godz_nj(2)	**20**

PODSUMOWANIE

- Python to język interpretowany o przejrzystej składni, za pomocą którego można pisać mniej i bardziej skomplikowane programy. Jego skrypty są analizowane przez interpreter, a następnie wykonywane przez komputer.
- Polecenia w języku Python grupowane są w bloki, które uzyskuje się przez wcięcia w wierszach (indentację).
- Język Python rozróżnia małe i wielkie litery.
- Rozwiązywany problem warto podzielić na mniejsze problemy, które można niezależnie rozwiązać przez definiowanie funkcji. Gdy problem jest prosty, wystarczy jedna funkcja.
- Definicję funkcji zaczyna się od słowa kluczowego **def**, następnie podaje się nazwę funkcji i wstawia nawiasy, w których można umieścić parametr. Deklarację należy zakończyć dwukropkiem. Kolejne wiersze kodu zawierają instrukcje do wykonania.
- Funkcje mogą przekazywać wynik (polecenie **return**) lub powodować określony skutek na ekranie (instrukcja **print**).
- Podstawowe operatory arytmetyczne i porównania są zbliżone do używanych w języku naturalnym, dlatego po napisaniu kilku programów korzystanie z nich staje się intuicyjne.

PYTANIA SPRAWDZAJĄCE

1. Jaka liczba będzie wynikiem operacji `5 / 2`, a jaka – wynikiem operacji `5 // 2`?
2. Czym się różni polecenie `print("a")` od `print(a)`?
3. Do czego służy polecenie `return` w funkcji?

ZADANIA DODATKOWE

Zadanie 1. Ile kosztuje?

Zdefiniuj funkcję **cena(f)**, której parametrem jest cena w dolarach funta danego produktu, a wynikiem cena w złotówkach kilograma tego produktu. Sprawdź w internecie aktualny kurs dolara oraz przelicznik jednostek masy (funt i kilogram).

Zadanie 2. Prędkość w skali Beauforta

Zdefiniuj funkcję **skala(v)**, której parametrem jest prędkość wiatru w węzłach, a wynikiem przeliczona prędkość w skali Beauforta.

Zadanie 3. Która godzina będzie w Warszawie?

Różnica czasu między Los Angeles a Warszawą wynosi dziewięć godzin – jeżeli w Los Angeles zegar wskazuje godzinę 21, to w Warszawie jest godzina 6 rano. Przeanalizuj dane w tabeli i zdefiniuj funkcję **godz_waw(g)**, której parametrem jest pełna godzina w Los Angeles, a wynikiem aktualna godzina w Warszawie.

Wywołanie funkcji	Wynik
`godz_waw(8)`	17
`godz_waw(21)`	6

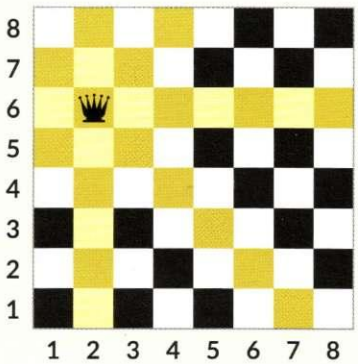
15. Definiowanie funkcji obliczeniowych

- Wykorzystanie instrukcji warunkowej w obliczeniach
- Instrukcja iteracji
- Analizowanie i testowanie rozwiązań

ZADANIE NA START

Ile pól atakuje hetman

Hetman (królowa) porusza się w tym samym rzędzie lub w tej samej kolumnie albo po przekątnych, w dowolnym kierunku, o dowolną liczbę niezajętych pól. Pola, do których figura może dotrzeć w jednym ruchu, są przez nią atakowane. Hetman stojący na szachownicy 8 × 8 na polu (2, 6), gdzie 2 oznacza numer kolumny, a 6 – numer wiersza, atakuje 14 + 9 = 23 pola. Określ liczbę atakowanych pól na szachownicy 32 × 32, gdy hetman znajduje się na polu (25, 30).



SZACHY, SZTUCZNA INTELIGENCJA I PROGRAMOWANIE

Najstarsza maszyna, która potrafiła w ograniczonym stopniu grać w szachy, powstała już pod koniec XIX w. Obecnie programiści tworzą coraz skuteczniejsze algorytmy, pozwalające maszynie zmieścić się w regulaminowym czasie rozgrywania partii szachowych, rozpatrujące pozycje, które wpłyną i nie wpłyną na ocenę sytuacji, wybierające kilka najbardziej obiecujących ruchów i znajdujące prawdopodobnie najlepszy ruch. W zależności od złożoności problemy szachowe mogą wymagać zastosowania prostej funkcji z parametrem i operatorów arytmetycznych albo bardziej zaawansowanych narzędzi.

POSZUKAJ W INTERNECIE

Poszukaj w internecie informacji na temat pojedynku Garriego Kasparowa ze stworzonym przez IBM systemem Deep Blue oraz innych pojedynków komputer vs. człowiek.

W tabeli przedstawiono podstawowe konstrukcje stosowane w języku Python.

Konstrukcja	Składnia	Przykład
Instrukcja przypisania	`nazwa_zmiennej = wartość`	`x = 3`
Instrukcja warunkowa prosta	`if warunek:` `  instrukcje`	`if x % 3 == 0:` `  print("liczba podzielna przez 3")`

Instrukcja warunkowa złożona	`if warunek:`
	`  instrukcje`
	`elif warunek:`
	`  instrukcje`
	`else:`
	`  instrukcje`
		`if x > 0:`
		`  print("liczba dodatnia")`
		`elif x == 0:`
		`  print("zero")`
		`else:`
		`  print("liczba ujemna")`

Pętla **for**	`for nazwa_zmiennej in sekwencja:` `  instrukcje`	`for i in range(10):` `  print(i)`

Pętla **while**	`while warunek:`
	`  instrukcje`
		`while x > 10:`
		`  print(x)`
		`  x = x - 1`

Funkcja	`def nazwa(parametry):` `  instrukcje`	`def pole_prostokata(bok1, bok2):` `  return bok1 * bok2`

Ćwiczenie 1. Ile pól ma szachownica

Szachownica złożona jest z równych pól, naprzemiennie białych i czarnych. Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile_pol(n)**, której parametrem jest rozmiar szachownicy, a wynikiem liczba pól na szachownicy. Parametr **n** może przyjmować wartości od 2 do 32.

Wywołanie funkcji	Wynik
ile_pol(8)	**64**
ile_pol(13)	**169**

- Plansza złożona jest z $n \times n$ pól, a więc $8 \cdot 8 = 64$, $13 \cdot 13 = 169$.
- Zdefiniuj funkcję **ile_pol(n)**.

```
1. def ile_pol(n):
2.     return n * n
```

- Sprawdź działanie skryptu z różnymi danymi.

Pamiętaj, aby następne ćwiczenia wykonywać w tym samym pliku.

INSTRUKCJA WARUNKOWA

Instrukcja warunkowa pozwala na wykonanie różnych obliczeń w zależności od tego, czy zdefiniowane wyrażenie logiczne jest prawdziwe, czy fałszywe.

- Jeśli liczba jest parzysta, podziel wartość przez 2, w przeciwnym przypadku odejmij 1.
- Jeśli liczba jest dodatnia, przypisz zmiennej znak 1, gdy ujemna przypisz –1, w przeciwnym przypadku przypisz 0.

Instrukcja warunkowa może składać się z jednej, dwóch lub trzech części.

- Na początku zawsze znajduje się słowo kluczowe **if** oraz warunek lub wyrażenie, które mogą mieć wartość **prawda** (1 lub inna wartość) lub **fałsz** (0). Wiersz należy zakończyć dwukropkiem.
- Potem następuje instrukcja lub ciąg instrukcji do wykonania w przypadku spełnienia warunku.
- Słowo kluczowe **elif** (skrót od *else if*) wprowadza nowy warunek lub wyrażenie oraz instrukcje do wykonania w przypadku spełnienia tego warunku.
- Słowo kluczowe **else** wprowadza instrukcję do wykonania, gdy żadne z powyższych warunków nie są prawdziwe.

ZAPAMIĘTAJ!

```
if <wyrażenie jest prawdziwe>:
     <wykonaj instrukcję 1>
elif <poprzednie wyrażenie jest fałszywe, ale to jest prawdziwe>:
     <wykonaj instrukcję 2>
else:
     <wykonaj instrukcję 3>
```

Instrukcja warunkowa na rys. 1 składa się z dwóch części. Można ją odczytać następująco:

- przypisz zmiennej **x** wartość 15;
- jeśli reszta z dzielenia całkowitego zmiennej **x** (liczby 15) przez 2 jest równa 0, wyświetl połowę tej liczby;
- w przeciwnym wypadku wyświetl liczbę o 1 mniejszą.

```
1. x = 15
2. if x % 2 == 0:
3.     x = x / 2
4. else:
5.     x = x - 1
```

Rys. 1. Przykładowa instrukcja warunkowa złożona z dwóch elementów

Instrukcja warunkowa na rys. 2 składa się z trzech części. Można ją odczytać następująco:

- przypisz zmiennej **x** wartość –12;
- jeśli **x** (liczba –12) jest większa od 0, przypisz zmiennej **zn** 1;
- jeśli poprzedni warunek jest fałszywy i **x** (liczba –12) jest mniejsze od 0, przypisz zmiennej **zn** –1;
- w przeciwnym wypadku przypisz zmiennej **zn** 0.

```
1. x = -12
2. if x > 0:
3.     zn = 1
4. elif x < 0:
5.     zn = -1
6. else:
7.     zn = 0
8. print(zn)
```

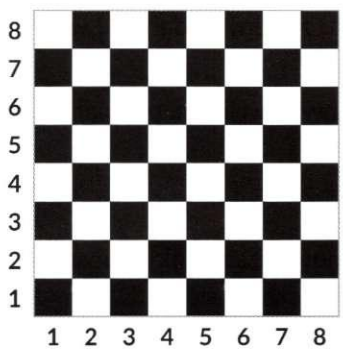
Rys. 2. Przykładowa instrukcja warunkowa złożona z trzech elementów

Ćwiczenie 2. Ile jest białych pól, a ile czarnych

Szachownica złożona jest z kwadratowych pól, naprzemiennie białych i czarnych. Lewy dolny róg (1, 1) zajmuje pole czarne. Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile_kolorowych(n, kolor)**, której parametrami są rozmiar szachownicy i kolor pola – **b** (białe) lub **c** (czarne) – a wynikiem jest liczba pól w danym kolorze. Parametr **n** może przyjmować wartości od 2 do 32.

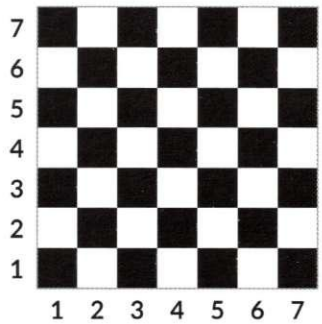
Wywołanie funkcji	Wynik
`ile_kolorowych(8, "b")`	32
`ile_kolorowych(7, "b")`	24
`ile_kolorowych(8, "c")`	32
`ile_kolorowych(7, "c")`	25

- W przypadku wartości parzystych parametru liczba pól w obu kolorach jest jednakowa. Aby wyznaczyć liczbę pól w danym kolorze, musisz pomnożyć przez siebie rozmiar szachownicy i podzielić przez 2: 8 * 8 // 2 = 32.



- W przypadku wartości nieparzystych parametru liczba pól białych jest o 1 mniejsza od liczby pól czarnych. Aby wyznaczyć liczbę pól w danym kolorze, musisz wprowadzić dwa warunki:
  - jeśli pola są białe, wynikiem będzie kwadrat rozmiaru szachownicy, podzielony przez 2: 7 * 7 // 2 = 24;
  - w przeciwnym wypadku (tj. dla pól czarnych) wynikiem będzie kwadrat rozmiaru szachownicy pomniejszony o 1, a następnie podzielony przez 2 i powiększony o 1.

  Powstaje pytanie: dlaczego? Zastanów się i postaraj się uzasadnić podany wzór.



- Zapisz odpowiednie formuły.
  - Liczba pól białych:

```
n * n // 2
```

  - Liczba pól czarnych:

```
(n * n - 1) // 2 + 1
```

- Zdefiniuj funkcję **ile_kolorowych(n, kolor)**.

```
1. def ile_kolorowych(n, kolor):
2.     if kolor == "b":
3.         return n * n // 2
4.     else:
5.         return (n * n - 1) // 2 + 1
```

- Sprawdź działanie skryptu z różnymi danymi, w tym z parzystym i nieparzystym rozmiarem szachownicy.

Jeżeli znasz liczbę pól szachownicy i liczbę pól białych, jak sprawdzić, czy dobrze jest policzona liczba pól czarnych?

PĘTLA FOR

Instrukcja iteracji służy do powtarzania jakiejś instrukcji lub bloku instrukcji w programie, np. dodawania wielu liczb do siebie. Dla każdego elementu sekwencji wykonywany jest podany ciąg instrukcji.

ZAPAMIĘTAJ!

```
for <nazwa zmiennej> in <sekwencja>:
    <instrukcje>
```

W przypadku pętli **for** zwyczajowo używa się zmiennej sterującej pętlą o nazwie **i**, a jako sekwencji iteratora **range(od, do, krok)**. Iterator **range()** może mieć jeden parametr „do" (koniec zakresu), dwa parametry „od" i „do" (początek i koniec zakresu) albo trzy parametry „od", „do" i „krok" (początek zakresu, koniec zakresu i krok, o jaki zwiększa się kolejne wartości), np.

- **range(7)** – definiuje zakres od 0 do 6;
- **range(3, 7)** – definiuje zakres od 3 do 6, bo ostatnia generowana wartość to największa liczba całkowita mniejsza niż ograniczenie górne;
- **range(1, 7, 2)** – definiuje liczby od 1, nie większe niż 7 i co 2, a więc: 1, 3 i 5.

Ćwiczenie 3. Zestawienie liczby pól białych i czarnych

Szachownica złożona jest z kwadratowych pól, naprzemiennie białych i czarnych. Lewy dolny róg (1, 1) zajmuje pole czarne. Efektem wywołania funkcji **testuj_szachownica(n)** powinno być wyświetlenie: kolejnej liczby, łącznej liczby pól szachownicy oraz liczby pól białych i czarnych dla wartości od 2 do 32. Przenalizuj poniższą funkcję, znajdź i popraw błędy.

```
1. def testuj_szachownica(n):
2.     for i in range(2, n):
3.         print (i, ile_pol(i, "b"), ile_kolorowych(i), ile_kolorowych("c", i))
```

Pierwsze wiersze zestawienia będą miały następujące wartości:

n	Liczba pól	Liczba pól białych	Liczba pól czarnych
2	4	2	2
3	9	4	5
4	16	8	8
5	25	12	13
6	36	18	18
7	49	24	25
8	64	32	32
9	81	40	41
10	100	50	50

- Skorzystaj z rozwiązań ćwiczeń 1 i 2. Zwróć uwagę na liczbę i kolejność parametrów. Funkcja **ile_pol(n)** powinna mieć jeden parametr, natomiast funkcja **ile_kolorowych(n, kolor)** – dwa parametry (pierwszy to rozmiar szachownicy, a drugi to kolor szukanych pól).

- Popraw błędy w kodzie funkcji.

```
1. def testuj_szachownica(n):
2.     for i in range(2, 33):
3.         print (i, ile_pol(i), ile_kolorowych(i, "b"), ile_kolorowych(i, "c"))
```

PRZYDATNE FUNKCJE

W języku Python istnieje wiele funkcji, które mogą ułatwić zapis algorytmów. Należą do nich m.in.

- funkcja **min()**, która wyznacza najmniejszą z wartości podanych jako argument – np. wynikiem funkcji **min(5, 2, 11)** jest 2;
- funkcja **max()**, która wyznacza największą z wartości podanych jako argument – np. wynikiem funkcji **max(5, 2, 11)** jest 11.

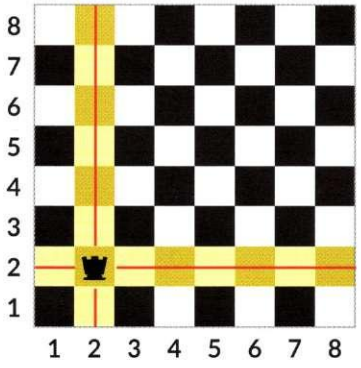
Pełny spis funkcji pomocniczych zawarty jest w dokumentacji języka na stronie **https://docs.python.org/3/library/functions.html**.

Ćwiczenie 4. Ile pól atakuje wieża

Wieża porusza się w tym samym rzędzie lub w tej samej kolumnie, w dowolnym kierunku, o dowolną liczbę pól. Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile_wieza(n, x, y)**, której parametrami są rozmiar szachownicy oraz położenie figury (**x**, **y**), a wynikiem jest liczba pól atakowanych przez wieżę. Parametr **n** może przyjmować wartości od 2 do 32. Parametry **x** i **y** mogą przyjmować wartości w przedziale od **1** do **n**.

Wywołanie funkcji	Wynik
ile_wieza(8, 2, 2)	**14**
ile_wieza(13, 3, 4)	**24**

- Rozważ liczbę pól atakowanych w pionie i poziomie na szachownicy o rozmiarze 8.



- Zdefiniuj funkcję **ile_wieza(n, x, y)**.

```
1. def ile_wieza(n, x, y):
2.     return 2 * (n - 1)
```

- Sprawdź działanie skryptu z różnymi danymi.

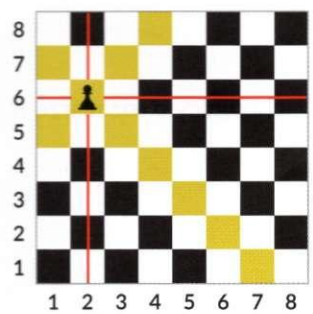
Ćwiczenie 5. Ile pól atakuje goniec

Goniec przesuwa się po przekątnych o dowolną liczbę pól i w dowolnym kierunku. Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile_goniec(n, x, y)**, której parametrami są rozmiar szachownicy oraz położenie figury (**x, y**), a wynikiem jest liczba pól atakowanych przez gońca. Parametr **n** może przyjmować wartości od 2 do 32. Parametry **x** i **y** mogą przyjmować wartości od **1** do **n**.

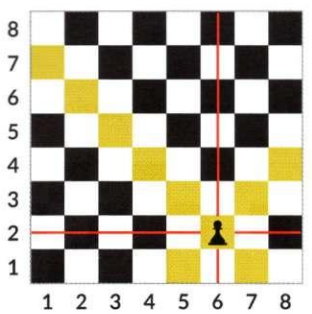
Wywołanie funkcji	Wynik
`ile_goniec(8, 2, 2)`	9
`ile_goniec(13, 3, 4)`	16

- W sposób naturalny położenie figury dzieli planszę na cztery części. Lewy górny róg, lewy dolny, prawy dolny róg i prawy górny. Rozważmy tę ostatnią część. Liczba pól atakowanych przez gońca o położeniu **(x, y)** to minimum z liczb **n - x** oraz **n - y**. W pozostałych częściach jest podobnie.

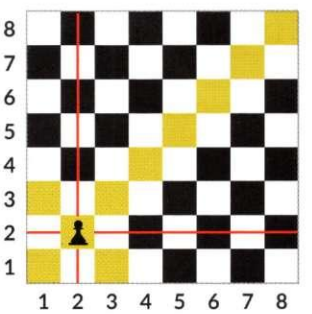
Goniec w prawej górnej części atakuje...



2 pola, gdy stoi na pozycji (2, 6)
min (8 - 2, 8 - 6) = 2



2 pola, gdy stoi na pozycji (6, 2)
min (8 - 6, 8 - 2) = 2



6 pól, gdy stoi na pozycji (2, 2)
min (8 - 2, 8 - 2) = 6

- Zapisz odpowiednie formuły.
  - I część:

```
min(x - 1, n - y)
```

  - II część:

```
min(x - 1, y - 1)
```

- III część:

```
min(n - x, y - 1)
```

- IV część:

```
min(n - x, n - y)
```

- Zdefiniuj funkcję **ile_goniec(n, x, y)**.

```
1. def ile_goniec(n, x, y):
2.     return min(x - 1, n - y) + min(x - 1, y - 1) + min(n - x, y - 1)
       + min(n - x, n - y)
```

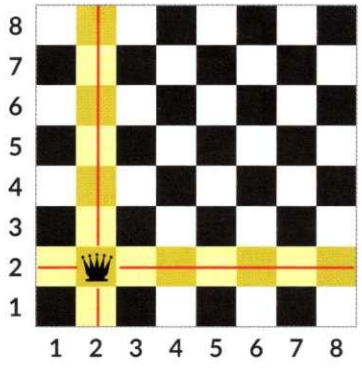
- Sprawdź działanie skryptu z różnymi danymi. Z jakimi danymi warto przetestować funkcję, aby się przekonać, że jest poprawnie napisana?

Ćwiczenie 6. Ile pól atakuje hetman

Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile_hetman(n, x, y)**, której parametrami są rozmiar szachownicy oraz położenie figury (**x, y**), a wynikiem jest liczba pól atakowanych przez hetmana. Parametr **n** może przyjmować wartości od 2 do 32. Parametry **x** i **y** mogą przyjmować wartość w przedziale od **1** do **n**.

Wywołanie funkcji	Wynik
ile_hetman(8, 2, 2)	**23**
ile_hetman(13, 3, 4)	**40**

- Rozważ liczbę pól atakowanych w pionie i poziomie na szachownicy o rozmiarze 8.



- Skorzystaj z rozwiązań ćwiczeń 4 i 5.

- Zdefiniuj funkcję **ile_hetman(n, x, y)**.

```
def ile_hetman(n, x, y):
    return ile_wieza(n, x, y) + ile_goniec(n, x, y)
```

- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 7. Zestawienie liczby pól atakowanych przez hetmana

Dla planszy o rozmiarze 8 wykonaj zestawienie liczby pól atakowanych przez hetmana w zależności od jego położenia.

	1	2	3	4	5	6	7	8
1	21	21	21	21	21	21	21	21
2	21	23	23	23	23	23	23	21
3	21	23	25	25	25	25	23	21
4	21	23	25	27	27	25	23	21
5	21	23	25	27	27	25	23	21
6	21	23	25	25	25	25	23	21
7	21	23	23	23	23	23	23	21
8	21	21	21	21	21	21	21	21

PODSUMOWANIE

- Ważną umiejętnością programistyczną jest zapis w języku formalnym tego, co można wyliczyć bez komputera.
- Podstawy pracy w języku tekstowym obejmują m.in. zapis algorytmu w postaci sekwencji poleceń, zapisywanie obliczeń, wykorzystanie instrukcji warunkowych i pętli oraz definiowanie funkcji.
- Instrukcja warunkowa pozwala na wykonanie różnych obliczeń w zależności od tego, czy zdefiniowane wyrażenie logiczne jest prawdziwe czy fałszywe. Może składać się z jednej, dwóch lub trzech części. Na początku znajduje się słowo kluczowe **if** oraz warunek lub wyrażenie, które mogą mieć wartość **prawda** (1 lub inna wartość) lub **fałsz** (0). Wiersz należy zakończyć dwukropkiem. Potem następuje instrukcja lub ciąg instrukcji do wykonania w przypadku spełnienia warunku. Słowo kluczowe **elif** wprowadza nowy warunek lub wyrażenie oraz instrukcje do wykonania w przypadku spełnienia tego warunku. Słowo kluczowe **else** wprowadza instrukcję do wykonania, gdy żaden z powyższych warunków nie jest prawdziwy.
- Instrukcja iteracji służy do powtarzania jakiejś instrukcji lub bloku instrukcji w programie, np. dodawania wielu liczb do siebie. Dla każdego elementu sekwencji wykonywany jest podany ciąg instrukcji. W przypadku pętli **for** zwykle używa się zmiennej sterującej pętlą o nazwie **i**, a jako sekwencji iteratora **range(od, do, krok)**.

- Funkcja **range()** może mieć jeden parametr „do" (koniec zakresu), dwa parametry „od" i „do" (początek i koniec zakresu) albo trzy parametry „od", „do" i „krok" (początek zakresu, koniec zakresu i krok, o jaki zwiększa się kolejne wartości).
- W języku Python są dostępne standardowe funkcje, choćby takie jak **min()** i **max()**, które odpowiednio wyznaczają najmniejszą lub największą z wartości podanych jako argument.

PYTANIA SPRAWDZAJĄCE

1. Czy w instrukcji warunkowej zawsze występuje słowo kluczowe **else**? Odpowiedź uzasadnij.
2. Jak w języku Python zapisać instrukcję iteracji, by zmienna sterująca pętlą zmieniała wartości od 10 do 0 włącznie?
3. Z jakimi wartościami parametrów warto testować każde zadanie?

ZADANIA DODATKOWE

Zadanie 1. Ile pól atakuje król

Król może się przesuwać pionowo, poziomo i po przekątnej, ale tylko o jedno pole. Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile_krol(n, x, y)**, której parametrami są rozmiar szachownicy oraz położenie figury (**x, y**), a wynikiem jest liczba pól atakowanych przez króla. Parametr **n** może przyjmować wartości od 2 do 30. Parametry **x** i **y** mogą przyjmować wartości od **1** do **n**.

Wywołanie funkcji	Wynik
`ile_krol(8, 1, 1)`	3
`ile_krol(10, 4, 4)`	8

Zadanie 2. Ile pól atakuje skoczek

Skoczek porusza się o dwa pola w przód i jedno w bok lub jedno w przód i dwa w bok. Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile_skoczek(n, x, y)**, której parametrami są rozmiar szachownicy oraz położenie figury (**x, y**), a wynikiem jest liczba pól atakowanych przez skoczka. Parametr **n** możne przyjmować wartości od 2 do 30. Parametry **x** i **y** mogą przyjmować wartości od **1** do **n**.

Wywołanie funkcji	Wynik
`ile_skoczek(8, 1, 1)`	2
`ile_skoczek(10, 4, 4)`	8

Zadanie 3. Czy można przejść z pola na pole daną figurą

Dostępna jest szachownica o rozmiarze **n** = 8. Przeanalizuj poniższe tabele i zdefiniuj funkcje **czy_wieza(xp, yp, xk, yk)**, **czy_goniec(xp, yp, xk, yk)** oraz **czy_hetman(xp, yp, xk, yk)**, których parametrami są położenie początkowe i końcowe figury, a wynikiem jest **True**, jeśli w jednym ruchu można przejść z położenia początkowego do końcowego daną figurą, lub **False**, jeśli nie da się tego zrobić. Parametry mogą przyjmować wartości od 1 do 8.

Wywołanie funkcji	Wynik
`czy_wieza(1, 1, 1, 4)`	`True`
`czy_wieza(2, 2, 7, 8)`	`False`

Wywołanie funkcji	Wynik
`czy_goniec(1, 1, 4, 4)`	`True`
`czy_goniec(2, 2, 7, 8)`	`False`

Wywołanie funkcji	Wynik
`czy_hetman(1, 1, 1, 4)`	`True`
`czy_hetman(2, 2, 7, 8)`	`False`

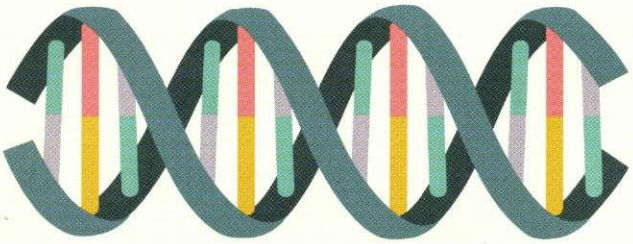
16. Wyszukiwanie wzorca w tekście

- Podstawowe operacje na napisach
- Algorytm naiwny wyszukiwania wzorca w tekście
- Porównywanie i przeszukiwanie napisów

ZADANIE NA START

Jakie symbole można znaleźć w DNA?

DNA to kwas deoksyrybonukleinowy. Tworzą go nukleotydy zbudowane z deoksyrybozy, jednej grupy fosforanowej oraz jednej zasady azotowej: adeniny (A), guaniny (G), cytozyny (C) lub tyminy (T). Przykładowy łańcuch DNA, który zawiera symbole A, C, G i T, to TCTAAAGATATCGGG. Długość tego łańcucha wynosi 15. Policz, ile razy symbole A, C, G i T występują w łańcuchu TCTAACAGCCCCATATCGGG.



PODSTAWOWE OPERACJE NA NAPISACH

W języku Python **napis** to ciąg znaków ujęty w podwójne lub pojedyncze cudzysłowy – nie tylko litery, lecz także cyfry czy znaki takie jak spacja.

Aby wyświetlić napis zdefiniowany jako zmienna, należy zastosować polecenie **`print`**.

ZAPAMIĘTAJ!

```
<zmienna> = "<ciąg znaków>"
print(<zmienna>)
```

Na rys. 1 pokazano przykład definiowania zmiennych, których wartością są napisy.

```
1. s1 = "biologia"
2. s2 = "molekularna"
3. print(s1)
4. print(s2)
```

Rys. 1. Definiowanie zmiennych

Napisy można łączyć ze sobą (jest to tzw. **konkatenacja**). W Pythonie łańcuchy znaków dodaje się za pomocą znaku +.

ZAPAMIĘTAJ!

```
<zmienna1> = "<ciąg znaków>"
<zmienna2> = "<ciąg znaków>"
print(<zmienna1> + <zmienna2>)
```

Oto przykład dodawania napisów. W wierszu czwartym dodawane są trzy napisy, środkowy napis (" ") jest spacją.

```
1. s1 = "biologia"
2. s2 = "molekularna"
3. print(s1 + s2)
4. print(s1 + " " + s2)
```

Rys. 2. Dodawanie napisów

WYZNACZANIE DŁUGOŚCI NAPISU

Wynikiem funkcji **len()** jest długość łańcucha znaków podanego jako jej argument.

ZAPAMIĘTAJ!

```
len(<zmienna>)
```

Oto deklaracja zmiennej **napis**, przypisanie jej wartości jako sumy trzech napisów (**s1**, spacja i **s2**), dodanie wartości funkcji obliczającej długość napisu (liczba znaków).

```
1. s1 = "biologia"
2. s2 = "molekularna"
3. napis = s1 + " " + s2
4. print(len(napis))
```

Rys. 3. Deklaracja zmiennej **napis**

ANALIZOWANIE NAPISU

Jeśli trzeba przeanalizować napis znak po znaku, można to zrobić za pomocą pętli **for**.

ZAPAMIĘTAJ!

```
for <znak> in <napis>:
    print(<znak>)
```

Oto pętla **for**, która pozwala przeanalizować zmienną **napis** znak po znaku.

```
1. napis = "genetyka"
2. for z in napis:
3.     print(z)
```

Rys. 4. Kolejne litery zmiennej **napis**

Pojedyncze znaki napisu można przeanalizować również z wykorzystaniem **indeksowania**. Umieszczona po nazwie zmiennej w nawiasie kwadratowym wartość indeksu wskazuje odpowiedni znak napisu – od zera. Jeśli zmiennej **napis** przypisze się wartość **"genetyka"**, to **napis[0]** będzie oznaczać **"g"**, **napis[4]** będzie oznaczać **"t"**, a **napis[7]** – **"a"**.

W języku Python istnieje również możliwość indeksowania od końca – wtedy zaczyna się od –1.

0	1	2	3	4	5	6	7
g	e	n	e	t	y	k	a
-8	-7	-6	-5	-4	-3	-2	-1

Drugi sposób analizowania litera po literze polega na zastosowaniu pętli **for** z funkcją **range**. Pętla jest wykonywana tyle razy, ile liter ma napis, a indeks zmienia swoje wartości od 0 do liczby (długość napisu – 1).

ZAPAMIĘTAJ!

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

Na rys. 5 pokazano pętlę **for** ze zmienną **napis** przedstawioną znak po znaku z wykorzystaniem indeksowania.

```
1. napis = "genetyka"
2. for i in range(len(napis)):
3.     print(napis[i])
```

Rys. 5. Kolejne litery zmiennej **napis** z indeksowaniem

W Pythonie, jak w każdym języku programowania, typ napisowy ma wiele wbudowanych funkcji, których stosowanie ułatwia pracę. Jedną z nich jest metoda **count()**, za pomocą której można sprawdzić, ile razy w napisie pojawia się dany znak lub ciąg znaków.

Poniżej przedstawiono wykorzystanie metody **count()**. Wywołuje się ją poprzez napisanie nazwy zmiennej, kropki i nazwy metody.

```
1. napis = "genetyka"
2. print(napis.count("e"))
```

Rys. 6. Liczba wystąpień znaku **"e"** w zmiennej **napis**

Ćwiczenie 1. Zliczanie pojedynczych znaków w tekście

Przeanalizuj dane w tabeli i zdefiniuj funkcję **ile(s)**, której parametrem jest ciąg znaków, a wynikiem lista czterech liczb całkowitych oznaczających, ile razy symbole A, C, G i T występują w tym ciągu znaków.

Wywołanie funkcji	Wynik
`ile("CAATAAAAA")`	`[7, 1, 0, 1]`
`ile("TCTAAAGATATCGGG")`	`[5, 2, 4, 4]`

- Przejrzyj dany ciąg znaków **"CAATAAAAA"** i policz, ile znajduje się w nim poszczególnych znaków.

Sposób 1

Algorytm wykorzystujący pętlę **for** i instrukcję warunkową

- Zapisz kolejne formuły.
  Aby zliczyć wystąpienia znaku A w napisie, należy utworzyć zmienną pomocniczą **ile_A**, która będzie przechowywała liczbę wystąpień symbolu A w danym napisie. Algorytm realizujący to zadanie polega na analizowaniu napisu znak po znaku. W przypadku napotkania litery A wartość zmiennej **ile_A** jest zwiększana o 1.

  ```
  ile_A += 1
  ```

  Ponieważ to samo dotyczy pozostałych liter, można zastosować instrukcję warunkową – napis jest złożony wyłącznie z liter ACGT, więc jeśli przetwarzaną literą nie jest A, C lub G, to jest nią litera T.

```
if z == "A":
    ile_A += 1
elif z == "C":
    ile_C += 1
elif z == "G":
    ile_G += 1
else:
    ile_T += 1
```

- Wynikiem funkcji ma być lista czterech liczb, dlatego należy je zapisać w nawiasach [ ].

```
[ile_A, ile_C, ile_G, ile_T]
```

- Zapisz algorytm – zdefiniuj funkcję **ile(s)**.

```
def ile(s):
    ile_A = 0; ile_C = 0
    ile_G = 0; ile_T = 0
    for z in s:
        if z == "A":
            ile_A += 1
        elif z == "C":
            ile_C += 1
        elif z == "G":
            ile_G += 1
        else:
            ile_T += 1
    return [ile_A, ile_C, ile_G, ile_T]
```

Sposób 2

Algorytm wykorzystujący operacje na napisach

- Zastosuj metodę **count()** – wykorzystaj **s.count("A")** do obliczenia liczby wystąpień symbolu A, **s.count("C")** do obliczenia liczby wystąpień symbolu C itd.
- Zapisz algorytm – zdefiniuj funkcję **ile(s)**.

```
def ile(s):
    return [s.count("A"), s.count("C"), s.count("G"), s.count("T")]
```

- Sprawdź działanie skryptu z różnymi danymi, w tym np. z napisami złożonymi z jednakowych liter.

Ćwiczenie 2. Zliczanie ciągów znaków w tekście

Przeanalizuj dane w tabeli i zdefiniuj funkcję **pary(s)**, której parametrem jest łańcuch DNA, a wynikiem liczba wystąpień par jednakowych symboli (np. TTAT - 1, TTTA - 2, TATA - 0).

Wywołanie funkcji	Wynik
pary("TGATTCTGAACAAGTGTT")	**[4]**
pary("TGATATTCGATGTGAAAAAGTCATACTGTT")	**[6]**

- Aby wyszukać jednakowe symbole, trzeba przeanalizować cały napis znak po znaku. Porównuje się pary, czyli bieżący znak z następnym, np. TG, GA, AT, TT itd. Pętla będzie wykonywana *x* razy, gdzie *x* to (długość napisu - 1).
- Zapisz kolejne formuły - przy konstruowaniu warunku zastosuj indeksowanie.

```
s[i] == s[i + 1]
```

- Zdefiniuj funkcję **pary(s)**.

```
1. def pary(s):
2.     ile = 0
3.     for i in range(len(s) - 1):
4.         if s[i] == s[i + 1]:
5.             ile += 1
6.     return ile
```

- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 3. Zmiana struktury ciągu znaków

Odwrotne uzupełnienie łańcucha DNA to ciąg utworzony przez odwrócenie danego ciągu i dopełnienie każdego symbolu, tj. zamiany A na T i odwrotnie oraz C na G i odwrotnie (np. CTGA → AGTC → TCAG). Przeanalizuj dane w tabeli i zdefiniuj funkcję **oduz(s)**, której parametrem jest łańcuch DNA, a wynikiem jego odwrotne uzupełnienie.

Wywołanie funkcji	Wynik
oduz("TGACCCA")	**"TGGGTCA"**
oduz("TGATATTCGATGTGAAAAA")	**"TTTTTCACATCGAATATCA"**

- Dla danego ciągu TGACCCA musisz utworzyć jego dopełnienie (zamiana parami liter) - wynikiem będzie ACTGGGT. Następnie trzeba go odwrócić, czyli zapisać od końca - wynikiem będzie TGGGTCA. Kolejność operacji (zamiana i odwracanie) nie ma znaczenia.

- Najpierw nadaj wartość początkową zmiennej **t**, która będzie wynikiem. Wynik ma być napisem, więc wartość początkowa będzie napisem pustym (**""**). Ponieważ w Pythonie nie da się zmodyfikować napisu – zmienić lub zastąpić w nim poszczególnych znaków – utwórz nową zmienną do przechowywania przekształconego łańcucha DNA, odwróć ciąg i zamień parami symbole, z uwzględnieniem kolejności napisów w dodawaniu.

POSZUKAJ W INTERNECIE

Dowiedz się, jak działa Rosalind – platforma do nauki bioinformatyki. Czy wiesz, jakie algorytmy leżą u podstaw filogenezy?

- Algorytm polega na przeglądaniu danego napisu. W trakcie przeglądania, gdy kolejnym symbolem będzie A, to zmiennej pomocniczej **zn** nada wartość T – i odwrotnie; gdy symbolem będzie C, to zmiennej pomocniczej **zn** nada wartość G – i odwrotnie. Do wyniku algorytm doda wartość zmiennej pomocniczej **zn** – w ten sposób odwróci przekształcany napis.

- Zdefiniuj funkcję **oduz(s)**.

```
1. def oduz(s):
2.     t = ""
3.     for i in range(len(s)):
4.         if s[i] == "A":
5.             zn = "T"
6.         elif s[i] == "T":
7.             zn = "A"
8.         elif s[i] == "C":
9.             zn = "G"
10.        else:
11.            zn = "C"
12.        t = zn + t
13.    return t
```

- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 4. Zmiana struktury ciągu znaków

Przeanalizuj dane w tabeli i zdefiniuj funkcję **ta(s)**, której parametrem jest łańcuch DNA, a wynikiem łańcuch DNA z * w miejscu symboli C, G i T.

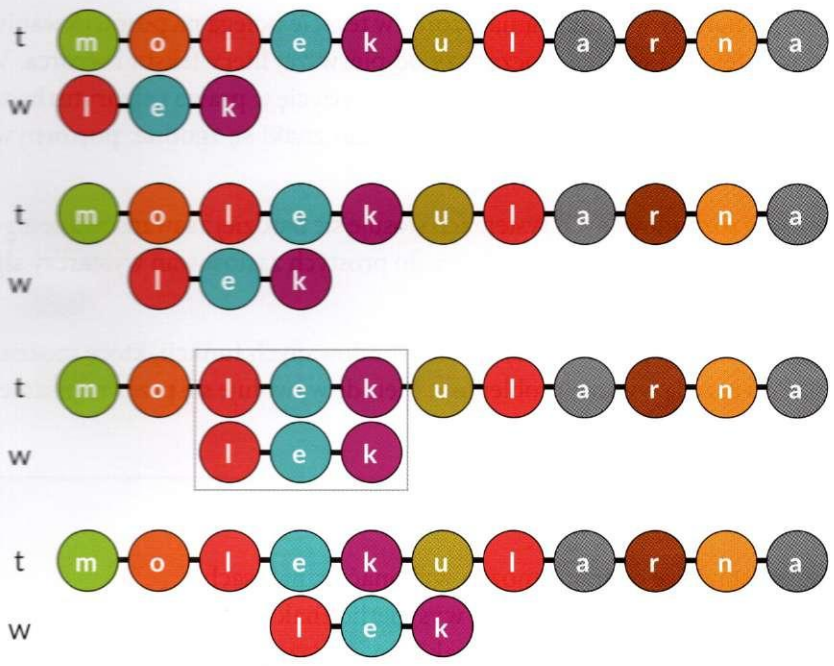
Wywołanie funkcji	Wynik
ta("TGACCCA")	**"**A***A"**
ta("TGATATTCGATGTGAAAAA")	**"**A*A****A****AAAAA"**

ALGORYTM NAIWNY WYSZUKIWANIA WZORCA W TEKŚCIE

Wzorzec to spójny ciąg znaków, który występuje w przeszukiwanym tekście – jest nim np. fraza **lek** występująca w słowie **molekularna**. Algorytmy umożliwiające wyszukiwanie wzorca w danym skończonym łańcuchu znaków wykorzystywane są m.in. w edytorach tekstu, przeglądarkach internetowych oraz programach pocztowych. Za pomocą wzorców można też określić, z czego ma się składać adres e-mail. Zadana do odszukania sekwencja znaków może się powtarzać albo w ogóle nie występować w analizowanym tekście.

ZAPAMIĘTAJ!

Algorytm naiwny wyszukiwania wzorca (inaczej algorytm N) to najprostszy algorytm, który polega na porównywaniu kolejnych liter tekstu (**t**) i wzorca (**w**), począwszy od pierwszej litery tekstu i wzorca. W razie niezgodności wzorzec przesuwa się o jedną pozycję w prawo i algorytm bada drugi znak tekstu i pierwszy znak wzorca. Jeśli znaki są zgodne, porównywane są następne znaki tekstu i wzorca.



Rys. 7. Cztery z dziewięciu etapów procesu wyszukiwania wzorca w słowie **molekularna**

Na powyższym rysunku wzorzec został odnaleziony na trzecim etapie procesu – wszystkie trzy litery są zgodne. Jeżeli chcesz znaleźć wszystkie wystąpienia wzorca w tekście, to kontynuuj przeszukiwanie tekstu od następnego znaku, aż do momentu, gdy liczba nieprzebadanych znaków jest mniejsza od długości wzorca.

WIEM WIĘCEJ

W edytorach tekstu można wyszukiwać konkretne napisy lub posłużyć się symbolami wieloznacznymi, np. użyć symbolu gwiazdki (*) do wyszukania ciągu znaków. Do wzorca b*a pasują takie wyrazy, jak biologia, biochemiczna, biofizyka, białka.

Ćwiczenie 5. Przeszukiwanie DNA

Dany jest łańcuch DNA o sekwencji TCTAACAGCCCCATATCGGG. Na którym etapie algorytmu N zostanie odnaleziony wzorzec CAT? Ile wzorców zostanie znalezionych?

PODSUMOWANIE

- Algorytmika i programowanie mają zastosowanie w różnych dziedzinach nauki, m.in. w biologii molekularnej, a także w codziennym funkcjonowaniu w społeczeństwie cyfrowym.
- Algorytm naiwny wyszukiwania wzorca w tekście polega na porównywaniu kolejnych liter tekstu i wzorca, począwszy od pierwszej litery tekstu i wzorca. W razie niezgodności wzorzec przesuwa się o jedną pozycję w prawo i algorytm bada drugi znak tekstu oraz pierwszy znak wzorca. Jeśli znaki są zgodne, porównywane są następne znaki tekstu i wzorca.
- Chociaż w rozbudowanych systemach stosuje się bardziej zaawansowane algorytmy wyszukiwania, które działają szybciej, do prostych zastosowań wystarczy algorytm naiwny.
- Napisy, czyli ciągi znaków, mają wiele wbudowanych funkcji, które można wykorzystać do rozwiązywania problemów. Metodę wywołuje się przez napisanie nazwy zmiennej, kropki i nazwy metody.

PYTANIA SPRAWDZAJĄCE

1. Jakie podstawowe operacje można wykonać na napisach?
2. W jaki sposób można w Pythonie wyświetlić znak po znaku? Podaj dwa sposoby.
3. Ile razy trzeba porównywać dwa znaki, aby znaleźć w tekście o długości **n** wzorzec o długości **w**?

ZADANIA DODATKOWE

Zadanie 1. Sekwencja błędów

Odległość Hamminga między dwoma łańcuchami DNA oznacza liczbę symboli, którymi różnią się oba łańcuchy. Odległość Hamminga między łańcuchami **s** i **t** o tej samej długości wynosi 8. Niepasujące symbole są zaznaczone kolorem:

- łańcuch **s** – CAGGCTACTACCGGTAT,
- łańcuch **t** – CATCGTAATGCAGGGCT.

Przeanalizuj dane w tabeli i zdefiniuj funkcję **hamming(s, t)**, gdzie **s** i **t** to łańcuchy o tej samej długości. Wynik funkcji to liczba będąca odległością Hamminga między danymi łańcuchami.

Wywołanie funkcji	Wynik
`hamming("CAGGTT", "CAGGTT")`	`0`
`hamming("CAGGCTACTACCGGTAT", "CATCGTAATGCAGGGCT")`	`8`

Zadanie 2. Jak wyszukać tekst?

Sprawdź, jak wyszukiwać napisy za pomocą symboli wieloznacznych w edytorze tekstu. Podaj kilka przykładów wyszukiwań.

17. Przetwarzanie napisów

- Wyodrębnianie fragmentu napisu
- Szyfry przestawieniowe
- Sprawdzanie, czy napis jest palindromem

ZADANIE NA START

Czy umiesz odczytać poniższe sentencje?

Odczytaj poniższe zdania, a następnie opisz metodę, jaką zastosowano w każdym przypadku.

patetyr śoin e wimra ęejzdneai
tytepa einśor w ęraim ainezdej
aqpdebtayity raoośynoiuel wy meixabrlęt jvewdfzgeinpiaaq
apeytt roinśe w mraię jndeieza

dgiz erdawr bąąit maw óiyrl ceą
eizdg awrd ąibąr mat yróiw ącel
godczaiseg dwrownad rląwbqiyąx tsaqmj wmicópriyh ltejcnąj
gidze dwra rbiąą tam wóriy lecą

eb zrpca yin eamk łocayz
zeb ycarp ein am yzcałok
bveuzm pqriapcoyl nbipek mhau kcoałlahctzmyz
bez parcy nie ma kłczoay

WYODRĘBNIANIE FRAGMENTU NAPISU

Podczas przetwarzania napisów można odwoływać się do poszczególnych znaków za pomocą indeksów. Umożliwia to wyodrębnianie fragmentów napisów. Indeksy to liczby z zakresu od 0 do (długość napisu – 1). Za pomocą nawiasów kwadratowych uzyskuje się dostęp do pojedynczego znaku – wystarczy podać jego indeks.

```
1. s = "szyfrowanie"
2. print(s[0])    # s
3. print(s[9])    # i
```

Rys. 1. Znak o indeksie 0 i 9

ZAPAMIĘTAJ!

W Pythonie jednowierszowy **komentarz** rozpoczyna się od znaku #. Komentarze wielowierszowe zawarte są między trzema cudzysłowami lub apostrofami.

Jeżeli w nawiasie kwadratowym poda się dwie liczby oddzielone dwukropkiem, uzyska się fragment napisu, zaczynając od indeksu, który jest pierwszą liczbą, kończąc na indeksie o 1 mniejszym niż druga liczba. Jeśli przed dwukropkiem nie będzie liczby, to domyślnie pierwsza liczba będzie równa 0.

ZAPAMIĘTAJ!

<napis>[<liczba1> : <liczba2>]
wyodrębni **<napis>** od indeksu **<liczba1>** do **<liczba2> - 1**

```
1. s = "szyfrowanie"
2. print(s[3:7])    # frow
3. print(s[0:5])    # szyfr
4. print(s[:5])     # szyfr
```

Rys. 2. Fragmenty napisu **s**

W nawiasach można też podać liczby ujemne. Dzięki temu znaki będą liczone od końca: –1 to wówczas ostatni znak, –2 przedostatni itd.

```
1. s = "szyfrowanie"
2. print(s[-1])       # e
3. print(s[-5:-2])    # wan
4. print(s[-5:])      # wanie
5. print(s[:-2])      # szyfrowan
```

Rys. 3. Fragmenty napisu **s** – indeksy ujemne

W prosty sposób można uzyskać napis od końca. Wystarczy podać w nawiasie kwadratowym po dwóch dwukropkach indeks –1.

```
1. s = "szyfrowanie"
2. print(s)         # szyfrowanie
3. print(s[::])     # szyfrowanie
4. print(s[::-1])   # einaworfyzs
```

Rys. 4. Napis normalnie i wspak

SZYFR PRZESTAWIENIOWY

Kodowanie wiadomości może się odbyć przez przestawienie kolejnych par znaków. W tak zaszyfrowanym tekście występują wszystkie znaki z tekstu jawnego, ale w innej kolejności. Kolejność liter zmieniana jest według określonego schematu – np. pisanie wspak.

ZAPAMIĘTAJ!

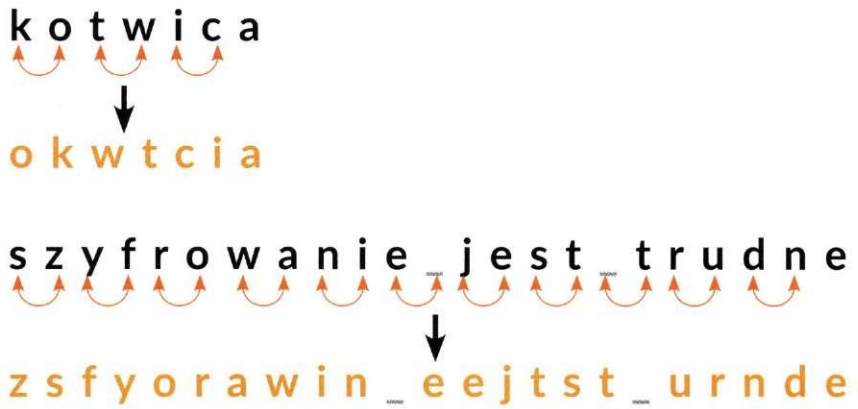
Szyfrogram to tekst, który został zaszyfrowany.
Tekst jawny to tekst, który nie został zaszyfrowany lub został odszyfrowany.
Nauka, która zajmuje się szyfrowaniem i deszyfrowaniem wiadomości, nosi nazwę kryptografii.

Ćwiczenie 1. Przestawianie parami

Przeanalizuj dane w tabeli i zdefiniuj funkcję **szyfr(s)**, której parametrem jest tekst jawny składający się z małych liter alfabetu łacińskiego i pojedynczych spacji, a wynikiem szyfrogram utworzony przez przestawienie kolejnych par znaków, przy czym spacja traktowana jest jak każda inna litera i również podlega zamianie.

Wywołanie funkcji	Wynik
`szyfr("kotwica")`	`"okwtcia"`
`szyfr("szyfrowanie jest trudne")`	`"zsfyorawin eejtst urnde"`

- Przeanalizuj algorytm krok po kroku.



- Do zmiennej **t** jest przypisany pusty napis. W każdym obrocie pętli rozpatrywane są dwa kolejne znaki, a dokładniej: zamieniana jest ich kolejność i po tej zamianie są one dopisywane na koniec zmiennej **t**. Gdy napis ma długość nieparzystą, dopisywany jest ostatni znak.

- Zapisz kolejne formuły.
  - Zastosuj pętlę **for**, aby przeglądać dany tekst co dwa znaki i dopisywać do zmiennej pary znaków zamienione kolejnością w stosunku do danych.

```
1. for i in range(0, len(s) - 1, 2):
2.     t = t + s[i + 1] + s[i]
```

  - Dopisanie ostatniego znaku to warunek wykonywany po zakończeniu pętli.

```
1. if len(s) % 2 != 0:
2.     t = t + s[len(s) - 1]
```

- Zdefiniuj funkcję **szyfr(s)**.

```
1. def szyfr(s):
2.     t = ""
3.     for i in range(0, len(s) - 1, 2):
4.         t = t + s[i + 1] + s[i]
5.     if len(s) % 2 != 0:
6.         t = t + s[len(s) - 1]
7.     return t
```

- Sprawdź działanie skryptu z różnymi danymi, w tym z napisami o parzystej i nieparzystej długości.

PARKAN

Parkan, który encyklopedia *Britannica* opisuje jako szyfr popularny w szkołach, jest prosty, ale osoba niemająca z nim wcześniej do czynienia musi się sporo nabiedzić, zanim odczyta ukrytą wiadomość, zwłaszcza że przy szyfrowaniu parkanem nie trzeba ograniczać się do dwóch rzędów. Prosta wersja szyfru polega na zapisaniu liter tekstu jawnego w dwóch wierszach. Na początek usuwa się wszystkie spacje – to częsta praktyka w kryptografii, ponieważ podział na wyrazy ułatwia złamanie szyfru. Następnie należy w pierwszym wierszu zapisać litery na pozycjach nieparzystych, a w drugim – na pozycjach parzystych. Aby utworzyć szyfrogram, trzeba odczytać tekst wierszami.

Tekst jawny	Szyfrowanie	Szyfrogram
albumy	a b m l u y	abmluy
domek	d m k o e	dmkoe

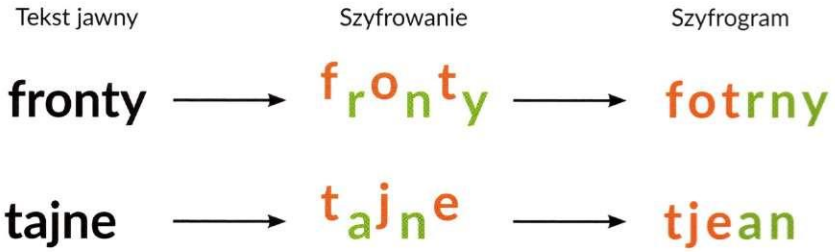
Rys. 5. Szyfrowanie parkanem

Ćwiczenie 2. Parkan

Przeanalizuj dane w tabeli i zdefiniuj funkcję **parkan(s)**, której parametrem jest tekst jawny złożony z małych liter alfabetu łacińskiego, a wynikiem – szyfrogram utworzony za pomocą szyfru parkan.

Wywołanie funkcji	Wynik
`parkan("fronty")`	`"fotrny"`
`parkan("tajne")`	`"tjean"`

- Przeanalizuj algorytm krok po kroku.



- Do pustego słowa najpierw dopisz znaki na indeksach parzystych, a potem nieparzystych.
- Zdefiniuj funkcję **parkan(s)**.

```
1. def parkan(s):
2.     w = ""
3.     for i in range(0, len(s), 2):
4.         w = w + s[i]
5.     for i in range(1, len(s), 2):
6.         w = w + s[i]
7.     return w
```

- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 3. Parkan bis

Zaproponuj krótsze rozwiązanie ćwiczenia 2.

PALINDROM

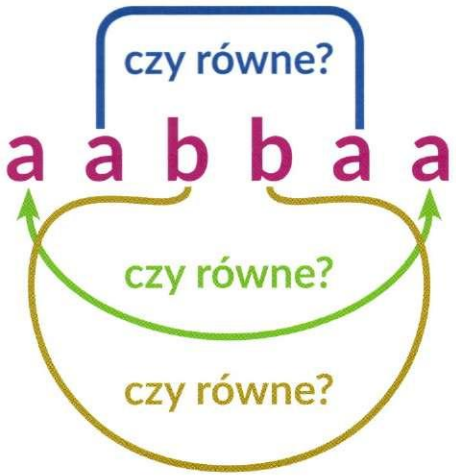
Palindrom (z gr. biec z powrotem) to słowo lub zdanie identyczne niezależnie od tego, czy jest czytane od lewej do prawej czy od prawej do lewej, np. kajak, potop, „A to kanapa pana Kota". Klasyczna metoda sprawdzania, czy tekst jest palindromem, polega na porównywaniu pierwszego znaku z ostatnim, drugiego znaku z przedostatnim itd.

Ćwiczenie 4. Czy to jest palindrom?

Przeanalizuj dane w tabeli i zdefiniuj funkcję logiczną **czy_palindrom(s)**, której parametrem jest napis zawierający tylko małe litery alfabetu łacińskiego, a wynikiem wartość logiczna **True**, gdy napis jest palindromem, lub **False**, gdy nim nie jest.

Wywołanie funkcji	Wynik
`czy_palindrom("aabbaa")`	`True`
`czy_palindrom("aaxyaa")`	`False`

- Przeanalizuj algorytm krok po kroku.



- Algorytm polega na sprawdzeniu znaków parami. Najpierw bada, czy pierwsza litera jest identyczna z ostatnią, a jeśli tak, to kontynuuje sprawdzanie kolejnych dwóch liter (drugiej i przedostatniej itd.). Jeśli litery nie są identyczne, kończy działanie funkcji i otrzymuje w wyniku **False**.
- Zdefiniuj funkcję **czy_palindrom(s)**.

```
1. def czy_palindrom(s):
2.     for i in range(len(s) // 2):
3.         if s[i] != s[-i - 1]:
4.             return False
5.     return True
```

- Sprawdź działanie funkcji z różnymi danymi, w szczególności z napisami, które są palindromami.

Ćwiczenie 5. Palindrom i odwracanie napisów

Rozwiąż ćwiczenie 4 z wykorzystaniem odwracania słowa.

Ćwiczenie 6. Czy zdanie jest palindromem?

Przeanalizuj dane w tabeli i zdefiniuj funkcję logiczną **czy_palindrom2(s)**, której parametrem jest napis zawierający małe litery alfabetu łacińskiego i spacje, a wynikiem wartość logiczna **True**, gdy napis jest palindromem, lub **False**, gdy nim nie jest.

Wywołanie funkcji	Wynik
`czy_palindrom2("a to kanapa pana kota")`	`True`
`czy_palindrom2("a to kanapka pana kota")`	`False`

- Zadanie można rozwiązać w dwóch krokach – najpierw pomiń spacje, potem zastosuj funkcję z ćwiczenia 4.
- Zdefiniuj funkcję **pomin(s)**, która usuwa spacje z tekstu.

```
1. def pomin(s):
2.     pom = ""
3.     for zn in s:
4.         if zn != " ":
5.             pom = pom + zn
6.     return pom
```

- Zdefiniuj funkcję **czy_palindrom2(s)** z wykorzystaniem funkcji **czy_palindrom(s)** z ćwiczenia 4.

```
1. def czy_palindrom2(s):
2.     return czy_palindrom(pomin(s))
```

- Sprawdź działanie skryptu z różnymi danymi, w szczególności z napisami, które zawierają spacje.

WIEM WIĘCEJ

Anagram to wyraz, wyrażenie lub zdanie powstałe przez przestawienie liter albo sylab innego wyrazu, wyrażenia lub zdania. W 1609 r. Galileusz odkrył pierścienie Saturna. Aby zabezpieczyć się przed wścibstwem żądnej sławy konkurencji, włoski astronom zanotował informację o swojej obserwacji za pomocą anagramu: SMAISMRMILMEPOETALEVMIBUNENUGTTAVIRAS. Oznaczało to: *Altissimum planetam tergeminum observavi* (Najwyższą planetę obserwowałem jako potrójną).

PRAKTYCZNE WYKORZYSTANIE SZYFROWANIA

Zabezpieczanie danych przez szyfrowanie to jedno z większych wyzwań w społeczeństwie cyfrowym. Szyfrować można m.in. pojedyncze pliki, foldery, dyski lub ich partycje, inne nośniki danych, e-maile albo transmisje danych. Możliwość szyfrowania oferują liczne aplikacje, a także popularne programy do kompresowania danych, takie jak 7-Zip czy WinRAR. Dzięki wykorzystaniu metod informatycznych wiele szyfrów można złamać, ale i tak odczytanie zaszyfrowanej wiadomości może sprawić trudność przeciętnemu człowiekowi.

POSZUKAJ W INTERNECIE

Wejdź na stronę **https://pl.khanacademy.org**, w dziale **Informatyka** wybierz *Podróż w krainę kryptografii* i dowiedz się, w jaki sposób ludzie chronili informacje kiedyś i jak robią to dziś.

PODSUMOWANIE

- W Pythonie można operować fragmentami napisów. Przez zastosowanie indeksów można wyodrębnić pojedynczą literę i cały fragment, a nawet odwrócić napis.
- Szyfrogram to tekst, który został zaszyfrowany. Tekst jawny to tekst, który nie został zaszyfrowany lub został odszyfrowany.
- Szyfry przestawieniowe polegają na zamianie kolejności liter, długość napisu nie ulega wówczas zmianie.
- Palindrom to słowo lub zdanie identyczne niezależnie od tego, czy jest czytane od lewej do prawej, czy od prawej do lewej. Sprawdzamy, czy napis jest palindromem, przez porównanie kolejno par liter: pierwszej i ostatniej, drugiej i przedostatniej itd. Jeżeli napotkamy różne znaki, to już wiemy, że napis nie jest palindromem.
- Zabezpieczanie danych przez szyfrowanie jest nieodzowne w społeczeństwie cyfrowym. Szyfrować można m.in. pojedyncze pliki, foldery, dyski lub ich partycje, e-maile oraz nośniki i transmisje danych.

PYTANIA SPRAWDZAJĄCE

1. Jak napisać pierwszy znak danego napisu? Jak ostatni?
2. W jaki sposób można napisać odwrócony napis?
3. Omówiony w ćwiczeniu 4 klasyczny sposób sprawdzania, czy napis jest palindromem, w wypadku napisu 10-literowego uruchomi pętlę pięć razy. Ile razy zostanie wywołana pętla gdy napis ma 11 liter?

ZADANIE DODATKOWE

Zadanie 1. Anagram

Przeanalizuj dane w tabeli i zdefiniuj funkcję **anagram(s)**, której parametrem będzie tekst jawny składający się z małych liter a, b lub c, a wynikiem napis, w którym wszystkie litery tekstu ustawione są w kolejności alfabetycznej.

Wywołanie funkcji	Wynik
anagram("acabbca")	**"aaabbcc"**
anagram("abccbaaaa")	**"aaaaabbcc"**

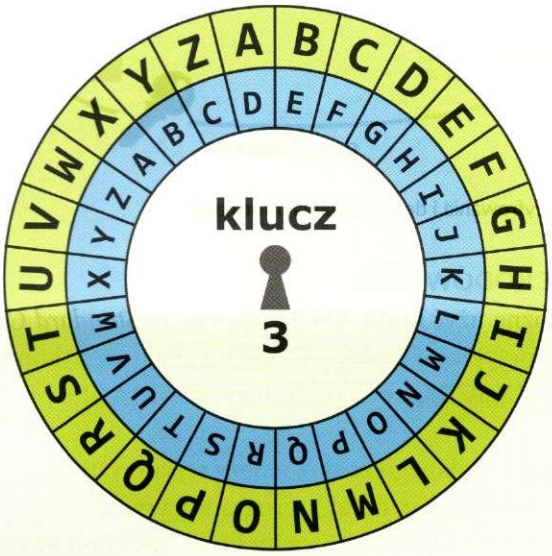
18. Szyfrowanie i deszyfrowanie tekstu

- Podstawowe pojęcia kryptograficzne
- Szyfr Cezara
- Szyfrowanie i odszyfrowywanie za pomocą kodów ASCII

ZADANIE NA START

Odczytaj wiadomość

Wiesz, że wiadomość **spotkanie o czwartej** zaszyfrowano za pomocą poniższego koła do szyfrowania jako **vsrwndqlh r fczduwhm**. Na tej podstawie odszyfruj wiadomość **vhnuhwqb olvw srg vfkrgdpl**.



SZYFR CEZARA

W stosowanym przez Juliusza Cezara **szyfrze podstawieniowym** każda litera tekstu jawnego zastępowana była inną według przyjętego klucza, znanego zarówno odbiorcy, jak i nadawcy wiadomości – i wykorzystywanego do szyfrowania oraz odszyfrowywania wiadomości.

ZAPAMIĘTAJ!

Klucz to informacja umożliwiająca wykonywanie pewnej czynności kryptograficznej, np. szyfrowania lub deszyfrowania.

Klucz w szyfrze Cezara to stała wartość, która wskazuje, o ile w prawo litera zaszyfrowana jest oddalona od pierwotnej. Dla danego napisu *t* i klucza *k* każdej literze napisu *t* przyporządkowuje się literę występującą o *k* miejsc dalej w alfabecie. Po dojściu do końca alfabetu należy zacząć liczyć od początku. Klucz 3 oznacza, że literze a odpowiada litera d, literze b – e, a literze z – c. Klucz 7 zaś oznacza, że literze a odpowiada litera h, literze b – i, a literze z – g. Rzymski wódz i polityk szyfrował listy do swoich przyjaciół za pomocą klucza 3.

Gdy ktoś ma tekst jawny i klucz, może uzyskać szyfrogram. Jedna metoda szyfrowania pozwala na uzyskanie różnych szyfrogramów dla różnych kluczy.

POSZUKAJ W INTERNECIE

Sprawdź, na czym polega steganografia.

Rys. 1. Proces szyfrowania i deszyfrowania informacji

WYKORZYSTANIE KODÓW ASCII

Do szyfrowania można użyć **kodu ASCII** (*American Standard Code for Information Interchange*), który przyporządkowuje każdemu znakowi w komputerze liczbę z zakresu 0–127 (np. kod ASCII litery a to 97, litery k – 107, litery z – 122).

Aby otrzymać kod ASCII danego znaku, należy wykorzystać funkcję **ord(znak)** – jeśli zastosuje się np. polecenie **ord("a")**, program wyświetli liczbę **97**, a jeśli poniższy skrypt – wyświetli litery i odpowiadające im kody ASCII od **97** do **103**.

```
1. for znak in "abcdefg":
2.     print(znak + ":", ord(znak))
```

Rys. 2. Skrypt pokazujący znaki podane w pętli **for** oraz odpowiadające im kody ASCII

Odwrotna funkcja to **chr(kod)**. Pozwala ona otrzymać znak odpowiadający danemu kodowi ASCII – jeśli zastosuje się np. polecenie **chr(97)**, program wyświetli literę **a**, a jeśli skrypt na stronie obok – litery **z**, **y**, **x**, **w**, **v**, **u**, **t**.

```
1. for kod in range(122, 115, -1):
2.     print(chr(kod))
```

Rys. 3. Skrypt umożliwiający wyświetlenie znaków na podstawie kodów ASCII podanych jako parametry w pętli **for**

Ćwiczenie 1. Szyfrowanie znaku

Przeanalizuj dane w tabeli i zdefiniuj funkcję **szyfruj_znak(znak, klucz)**, której parametrami są odpowiednio dowolna mała litera alfabetu łacińskiego oraz liczba całkowita nieujemna mniejsza od 1000, a wynikiem jest zaszyfrowana danym kluczem litera.

Wywołanie funkcji	Wynik
`szyfruj_znak("a", 3)`	`"d"`
`szyfruj_znak("z", 7)`	`"g"`

- Aby otrzymać pozycję danej litery w alfabecie, zamień literę na kod ASCII, a następnie odejmij od niej kod ASCII litery a (**97**). Do otrzymanej w ten sposób liczby dodaj klucz, a następnie znajdź resztę z dzielenia przez **26** (długość alfabetu). Do wyniku dodaj kod ASCII litery a (**97**) i zamień tak uzyskany kod ASCII na literę.

- Zapisz odpowiednią formułę.

```
chr((ord(dana_litera) - 97 + klucz) % 26 + 97)
```

- Zdefiniuj funkcję **szyfruj_znak(znak, klucz)**.

```
1. def szyfruj_znak(znak, klucz):
2.     return chr((ord(znak) - 97 + klucz) % 26 + 97)
```

- Przetestuj działanie skryptu z różnymi danymi. Jaki klucz trzeba podać, by zaszyfrować tekst tym samym znakiem, który pojawia się w tekście jawnym?

Ćwiczenie 2. Szyfrowanie tekstu

Przeanalizuj dane w tabeli i zdefiniuj funkcję **szyfruj(tekst, klucz)**, której parametrami są odpowiednio ciąg małych liter alfabetu łacińskiego oraz liczba całkowita nieujemna mniejsza od 1000, a wynikiem jest tekst zaszyfrowany danym kluczem.

Wywołanie funkcji	Wynik
`szyfruj("kiedyspotkanie", 13)`	`"xvrqlfcbgxnavr"`
`szyfruj("tajne", 133)`	`"wdmqh"`

Utwórz zmienną pomocniczą **pom**, która będzie stanowić napis pusty, a następnie dopisz do niej każdą kolejną zaszyfrowaną literę tekstu, np. dla **kiedyspotkanie** będą to kolejno litery **x**, **v**, **r**, ... **r**. Oznacza to, że wartości zmiennej **pom** to: **"x"**, **"xv"**, **"xvr"**, ..., **"xvrqlfcbgxnavr"**. W ten sposób wynikiem funkcji będzie wartość zmiennej **pom**.

- Zapisz funkcję **szyfruj(tekst, klucz)**.

```
1. def szyfruj(tekst, klucz):
2.     pom = ""
3.     for zn in tekst:
4.         pom = pom + szyfruj_znak(zn, klucz)
5.     return pom
```

- Sprawdź działanie skryptu z różnymi danymi. Zastanów się, jak za pomocą tej funkcji odszyfrować tekst.

Ćwiczenie 3. Odszyfrowywanie tekstu

Wynikiem zdefiniowanej poniżej funkcji **deszyfruj(tekst, klucz)** powinien być odszyfrowany tekst z danym kluczem.

```
1. def deszyfruj(tekst, klucz):
2.     return szyfruj(tekst, 26 + klucz)
```

Przeanalizuj dane w tabeli i popraw błąd w skrypcie.

Wywołanie funkcji	Wynik
deszyfruj("xvrqlfcbgxnavr", 13)	**"kiedyspotkanie"**
deszyfruj("wdmqh", 133)	**"tajne"**

- Odszyfrowanie tekstu to w istocie ponowne zaszyfrowanie z kluczem będącym jego dopełnieniem do 26. Jeśli tekst jawny zaszyfrowano z kluczem 13, to odszyfrowanie polega na zaszyfrowaniu kryptogramu kluczem 26 - 13 = 13 (co oznacza, że należy przesunąć się do przodu o 13 liter).

WIEM WIĘCEJ

Szyfr Cezara stosunkowo łatwo złamać, czyli odczytać wiadomość bez znajomości klucza. Chociaż wartość klucza jest nieograniczona, to praktycznie istnieje tylko 26 możliwości. Automatyczna technika łamania polega na analizie częstości liter. Jeżeli tekst jest dostatecznie długi i wiadomo, w jakim napisano go języku, można zbadać częstość występowania liter w szyfrogramie i porównać ją z częstością występowania liter w danym języku. Najlepsze dopasowanie z dużym prawdopodobieństwem wskaże klucz.

Ćwiczenie 4. Inaczej na pozycjach nieparzystych i parzystych

Przeanalizuj dane w tabeli i zdefiniuj funkcję **szyfruj2(tekst, klucz1, klucz2)**, której parametrami są odpowiednio ciąg małych liter alfabetu łacińskiego oraz dwie liczby całkowite nieujemne mniejsze od 1000, a wynikiem jest tekst zaszyfrowany

w ten sposób, że litery na pozycjach nieparzystych zaszyfrowano kluczem **klucz1**, a na parzystych – kluczem **klucz2**.

Wywołanie funkcji	Wynik
`szyfruj2("poufnytekst", 3, 10)`	`"zreixbdhuvd"`
`szyfruj2("poczekajnamnie", 123, 5)`	`"uhhsjdfcstrgnx"`

ZASTOSOWANIE SZYFROWANIA

Dzięki wykorzystaniu komputera szyfrowanie i deszyfrowanie przebiega szybciej, niż gdyby szyfrowało się informację samodzielnie. Łatwiej jest jednak złamać szyfr komputerowy, chociażby dlatego, że można wówczas wielokrotnie wykonać algorytm odszyfrowywania. Dlatego w praktyce stosuje się bardziej skomplikowane **metody szyfrowania, oparte na kryptografii asymetrycznej**. Zakłada ona istnienie dwóch kluczy: prywatnego, który zna tylko jego właściciel, i publicznego (jawnego), przy czym klucza prywatnego nie da się łatwo odtworzyć na podstawie publicznego. Szyfry asymetryczne wykorzystuje się m.in. w celu zabezpieczenia dostępu do zasobów hasłem lub cyfrowego podpisywania dokumentów.

PODSUMOWANIE

- W szyfrze Cezara do szyfrowania i deszyfrowania wiadomości stosuje się klucz, który wskazuje, o ile pozycji należy przesunąć literę.
- Klucz to informacja umożliwiająca wykonywanie pewnej czynności kryptograficznej, np. szyfrowania lub deszyfrowania. Gdy ktoś ma tekst jawny i klucz, może uzyskać szyfrogram. Jedna metoda szyfrowania pozwala na uzyskanie różnych szyfrogramów dla różnych kluczy.
- Do szyfrowania tekstów złożonych z liter alfabetu łacińskiego warto wykorzystywać kody ASCII.
- Obecnie w kryptografii stosuje się szyfry asymetryczne oparte na dwóch kluczach – prywatnym i publicznym.

PYTANIA SPRAWDZAJĄCE

1. Ile jest różnych kluczy w szyfrze Cezara?
2. Jakie wartości kluczy powodują, że tekst jawny i szyfrogram są identyczne?
3. Gdzie obecnie stosuje się elementy kryptografii?

ZADANIA DODATKOWE

Zadanie 1. Jakim kluczem zaszyfrowano?

Przeanalizuj dane w tabeli i zdefiniuj funkcję **jaki(tekst_jawny, szyfrogram)**, której parametrami są ciągi małych liter alfabetu łacińskiego, a wynikiem jest klucz z przedziału **[0,25]**, pozwalający zaszyfrować tekst jawny za pomocą metody Cezara, lub wartość **-1**, gdy szyfrogram nie mógł powstać z tekstu jawnego w wyniku szyfrowania metodą Cezara.

Wywołanie funkcji	Wynik
`jaki("niespodzianka", "pkgurqfbkcpmc")`	`2`
`jaki("zmiana", "anjbxb")`	`-1`

Zadanie 2. Szyfrowanie polskiego tekstu

Przeanalizuj dane w tabeli i zdefiniuj funkcję **szyfrujpl(tekst, klucz)**, której parametrami są odpowiednio ciąg małych liter alfabetu (mogą w nim występować litery typowo polskie) oraz liczba całkowita nieujemna mniejsza od 1000, a wynikiem jest zaszyfrowany tekst z danym kluczem.

Wywołanie funkcji	Wynik
`szyfrujpl("przekaż", 7)`	`"yzćkóęe"`
`szyfrujpl("wiadomość", 3)`	`"źlcfrorwę"`

Wskazówka: zdefiniuj zmienną **alfabet**, będącą napisem złożonym ze wszystkich liter alfabetu polskiego; pozycja litery w napisie odpowiada kodowi litery.

Zadanie 3. Złożenie szyfrów

Zaproponuj metodę szyfrowania opartą na złożeniu dwóch szyfrów, czyli zastosowaniu jednego szyfru, a potem drugiego do otrzymanego szyfrogramu. Napisz funkcję szyfrującą i deszyfrującą.