Szyfr Playfair w języku Python

- Wprowadzenie
- Przeczytaj
- Prezentacja multimedialna
- Sprawdź się
- Dla nauczyciela

Szyfr Playfair w języku Python

Źródło: Diomari Madulara, domena publiczna.

Szyfr Playfair jest jednym z bardziej popularnych szyfrów podstawieniowych, czyli polegających na zamianie poszczególnych znaków na inne. W przeciwieństwie jednak do szyfru Cezara, jest on poligramowy – nie zamieniamy pojedynczych znaków, lecz ich pary. Wprowadza to więc dodatkową warstwę obliczeń, które muszą zostać wykonane, aby zrealizować ten szyfr. Ogólną zasadę jego działania poznaliśmy w e-materiale Szyfr Playfair. Teraz zajmiemy się jego implementacją w języku Python.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Szyfr Playfair w języku Java,
- Szyfr Playfair w języku C++.

Więcej zadań? Przejdź do: Szyfr Playfair – zadania maturalne.

> **Twoje cele**
>
> - Przeanalizujesz zasadę działania szyfru Playfair.
> - Przygotujesz funkcje realizujące szyfrowanie oraz deszyfrowanie metodą Playfair.
> - Zaimplementujesz w języku Python program szyfrujący metodą Playfair.

Przeczytaj

Tworzenie szyfrogramu za pomocą szyfru Playfair polega na podzieleniu tekstu jawnego na grupy dwuznakowe, a następnie zastąpieniu par znaków/liter tekstu jawnego inną parą liter.

Do szyfrowania konieczna jest tabela tajnego słowa klucza – ma ona wymiary 5 x 5 i zawiera 25 liter alfabetu łacińskiego (alfabet łaciński ma 26 liter, więc opuszczamy jeden z rzadkich znaków – **x** lub **q** – lub traktujemy **i** oraz **j** jako jedną literę).

Oto przykładowa tabela sekretnego słowa klucza, utworzona ze słowa `szyfrogram`. Opuszczamy w kluczu wszystkie powtarzające się litery, a pozostałe pozycje w tabeli wypełniamy niewykorzystanymi literami alfabetu, zaczynając od litery **a** (uwaga: pomijamy literę **q**).

> **Przykład 1**
>
> | Przykładowa tabela sekretnego słowa | | | | |
> |---|---|---|---|---|
> | s | z | y | f | r |
> | o | g | a | m | b |
> | c | d | e | h | i |
> | j | k | l | n | p |
> | t | u | v | w | x |

Przyjrzyjmy się funkcji `sekretna_tablica()`, która będzie odpowiadać za stworzenie tabeli sekretnego słowa. W zmiennej `alfabet` znajduje się ciąg znaków, w którym przechowywane są litery alfabetu. Najpierw do tablicy wynikowej `tablica` wpisywane są po kolei unikalne znaki ze słowa `klucz`. Następnie wypełniamy tablicę kolejnymi znakami alfabetu, które nie występowały w słowie kluczowym.

```
1  def sekretna_tablica(klucz: str):
2      # dla tablicy 5 x 5 używamy alfabetu bez znaku "Q"
3      alfabet = "ABCDEFGHIJKLMNOPRSTUVWXYZ"
4      tablica = []
5
6      # kopiujemy znaki, ignorujemy duplikaty
7      for znak in klucz.upper():
8          if znak not in tablica and znak in alfabet:
9              tablica.append(znak)
10
```

```
    # wypełniamy pozostałe znaki w tablicy

    for znak in alfabet:
        if znak not in tablica:
            tablica.append(znak)
    return tablica
```

Przykład 2

Przykład wywołania funkcji `sekretna_tablica()`.

```
print(sekretna_tablica("szyfrogram"))
```

Wynik wywołania:

```
# ['S', 'Z', 'Y', 'F', 'R', 'O', 'G', 'A', 'M', 'B', 'C', 'D',
```

Otrzymany wynik możemy przekształcić tak, by odpowiadał zaprezentowanej przykładowej tabeli:

```
# ['S', 'Z', 'Y', 'F', 'R',
#  'O', 'G', 'A', 'M', 'B',
#  'C', 'D', 'E', 'H', 'I',
#  'J', 'K', 'L', 'N', 'P',
#  'T', 'U', 'V', 'W', 'X']
```

W kolejnym kroku dzielimy tekst jawny na części zawierające po dwie kolejne litery. Jeśli długość tekstu jawnego jest nieparzysta, dodajemy na końcu rzadki znak (np. `X`) po to, aby ostatnia część stanowiła parę znaków.

Możemy zdefiniować funkcję, która zrealizuje powyższe czynności i będzie zwracać kolejne części ciągu, tj. po dwa znaki na raz. Wykorzystamy do tego celu m.in.:

- pętlę `for ... in` do generowania kolejnych indeksów elementów ciągu,
- wyrażenie `tekst_jawny[i:i+2]` wykorzystujące notację indeksową do wyznaczania kolejnych dwuznakowych części ciągu,
- wyrażenie `yield`, które będzie zwracać kolejne części, po jednej na raz.

```
def zwroc_pary(tekst_jawny: str):
```

```
    # sprawdzamy długość tekstu
    if len(tekst_jawny) % 2 == 1:
        tekst_jawny += "x"
    tekst_jawny = tekst_jawny.upper()

    for i in range(0, len(tekst_jawny), 2):
        # zwracamy za pomocą yield kolejne pary znaków
        yield tekst_jawny[i:i+2]
```

Przykład 3

Przykład wywołania funkcji `zwroc_pary()`.

```
# przykład użycia funkcji w pętli:
for para in zwroc_pary("linuxtosystem"):
    print(f"kolejna para to: {para}")

# wyjście
# kolejna para to: ('L', 'I')
# kolejna para to: ('N', 'U')
# kolejna para to: ('X', 'T')
# kolejna para to: ('O', 'S')
# kolejna para to: ('Y', 'S')
# kolejna para to: ('T', 'E')
# kolejna para to: ('M', 'X')
```

Zgodnie z algorytmem znajdujemy obie litery z każdej pary w tabeli i podmieniamy je na litery według różnych schematów w zależności od warunku:

- **Warunek 1**: Obie litery znajdują się w tej samej kolumnie (np. `s` i `c`) – wówczas szyfrowane litery zostają zastąpione literami znajdującymi się w tej samej kolumnie, poniżej jawnych; jeśli jawna jest ostatnia w danej kolumnie, wówczas daną literę zastępujemy pierwszą literę z tej samej kolumny.

Przykład 4

Tekst jawny: "`sc`"

`s -> o, c -> j`

Tekst zakodowany: "`oj`"

s	z	y	f	r
o	g	a	m	b
c	d	e	h	i
j	k	l	n	p
t	u	v	w	x

Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

- **Warunek 2**: Obie litery znajdują się w tym samym wierszu (np. `s` i `y`) – wówczas szyfrowane litery zostają zastąpione literami znajdującymi się w tym samym wierszu, na prawo od jawnych; jeśli jawna jest ostatnia w danym wierszu, wówczas bierzemy do szyfrogramu pierwszą literę z tego samego wiersza.

Przykład 5

Tekst jawny: `"sy"`

```
s -> z, y -> f
```

Tekst zakodowany: `"zf"`

s	z	y	f	r
o	g	a	m	b
c	d	e	h	i
j	k	l	n	p
t	u	v	w	x

Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

- **Warunek 3**: Obie litery znajdują się w innym wierszu i innej kolumnie (np. `z` i `n`). Aby zaszyfrować daną literę, musimy znaleźć komórkę przecięcia wiersza, w którym znajduje się nasza szyfrowana litera oraz kolumnę, w której jest ulokowana druga litera.

Przykład 6

Tekst jawny: "`zn`"

`z -> f, n -> k`

Tekst zakodowany: "`fk`"

s	z	y	f	r
o	g	a	m	b
c	d	e	h	i
j	k	l	n	p
t	u	v	w	x

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Jak wynika z powyższych warunków, naszą sekretną tablicę, mimo że jest jednowymiarowa, musimy potraktować jako **dwuwymiarową**, aby określać położenie każdego znaku w wierszu i kolumnie.

Przykład 7

Na początku spróbujemy przedstawić jednowymiarową listę jako dwuwymiarową tablicę. W tym celu podzielimy listę na pięcioelementowe wiersze. Po wypisaniu pięciu elementów (znaków) w jednym wierszu użyjemy funkcji `print()`, by przejść do nowego wiersza. Następnie wypiszemy indeksy znaków rozpoczynających kolejne wiersze tabeli.

```
lista = ['S', 'Z', 'Y', 'F', 'R', 'O', 'G', 'A', 'M', 'B', 'C',
liczba_kolumn = 5
for i in range(len(lista)):
    if (i != 0 and divmod(i, liczba_kolumn)[1] == 0):
        print(i)
    print(lista[i], end=' ')
```

```
7
8 print()
9 print(lista.index('S'))
10 print(lista.index('O'))
11 print(lista.index('C'))
12 print(lista.index('J'))
13 print(lista.index('T'))
```

Po wykonaniu powyższego kodu (np. w trybie interaktywnym) otrzymamy następujący wynik:

```
1 S Z Y F R
2 O G A M B
3 C D E H I
4 J K L N P
5 T U V W X
6
7 0
8 5
9 10
10 15
11 20
```

Użyta tu funkcja `divmod()` zwraca tuplę (krotkę) zawierającą wynik dzielenia całkowitego oraz resztę z dzielenia dwóch argumentów. Kod `divmod(i, liczba_kolumn)[1]` pozwala odczytać resztę z dzielenia indeksu kolejnego znaku oraz liczby kolumn. Jeżeli reszta jest równa zeru, oznacza to, że wydrukowaliśmy pięcioznakowy wiersz i należy przejść do nowego wiersza, aby drukować następny.

Dzięki metodzie `index()` jesteśmy w stanie ustalić pozycję podanego elementu (znaku) w liście. Jak widać początkowe indeksy znaków rozpoczynających kolejne wiersze są wielokrotnościami liczby kolumn.

Zastanówmy się teraz, w jaki sposób możemy określić położenie dowolnego znaku w wierszu i kolumnie, jeżeli znamy jego indeks w liście oraz przyjmujemy, że **numery wierszy i kolumn zaczynają się od zera**.

Przykład 8

W trybie interaktywnym wykonajmy podane polecenia:

```
1 lista = ['S', 'Z', 'Y', 'F', 'R', 'O', 'G', 'A', 'M', 'B', 'C',
```

```
liczba_kolumn = 5
znak_1 = 'O'
print(lista.index(znak_1))
print(divmod(lista.index(znak_1), liczba_kolumn))
znak_2 = 'G'
print(lista.index(znak_2))
print(divmod(lista.index(znak_2), liczba_kolumn))
znak_3 = 'L'
print(lista.index(znak_3))
print(divmod(lista.index(znak_3), liczba_kolumn))
```

Po wykonaniu powyższego kodu otrzymamy wynik:

```
5
(1, 0)
6
(1, 1)
17
(3, 2)
```

W podanym przykładzie znak `'O'` ma indeks `5`, który podzielony przez liczbę kolumn daje wynik całkowity `1` i resztę `0` – co oznacza, że znak `'O'` znajduje się w wierszu `1` i kolumnie `0`. Analogiczne operacje na dwóch kolejnych znakach pozwalają ustalić, że `'G'` znajduje się w wierszu `1` i kolumnie `1`, a `'L'` w wierszu `3` i kolumnie `2`.

Teraz możemy przygotować kod generujący zaszyfrowane znaki, tzn. funkcję **szyfrującą**, która:

- otrzyma sekretną tablicę jako listę oraz parę znaków w tupli,
- ustali położenie przekazanych znaków w wierszach i kolumnach sekretnej tabeli,
- na podstawie omówionych warunków zastąpi je odpowiednimi znakami,
- zwróci zaszyfrowany ciąg (parę znaków) zapisany w zmiennej `tekst`.

W funkcji zostanie wykorzystany mechanizm **rozpakowywania tupli**, polegający na przypisywaniu wartości elementów tupli do pojedynczych zmiennych, np.:
`znak_1, znak_2 = para_znakow` lub
`wiersz_1, kolumna_1 = divmod(tablica.index(znak_1), 5)`.

Dla uproszczenia kodu liczbę kolumn zakodujemy za pomocą konkretnej wartości, czyli liczby `5`.

```
def znajdz_szyfr_pary(tablica: list, para_znakow: tuple):
```

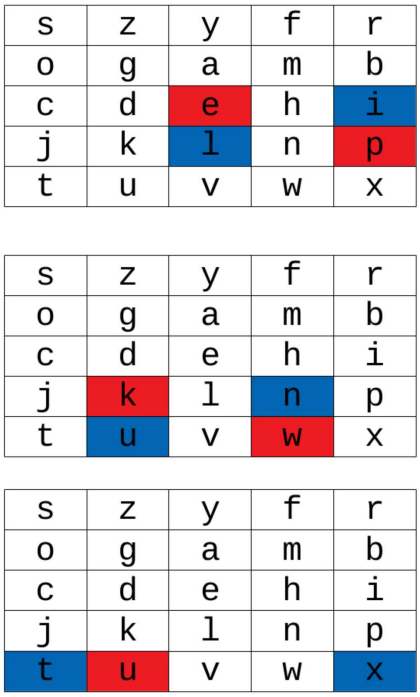
```
    znak_1, znak_2 = para_znakow
    wiersz_1, kolumna_1 = divmod(tablica.index(znak_1), 5)
    wiersz_2, kolumna_2 = divmod(tablica.index(znak_2), 5)

    if wiersz_1 == wiersz_2:
        tekst = tablica[wiersz_1 * 5 + (kolumna_1 + 1) % 5]
        tekst += tablica[wiersz_2 * 5 + (kolumna_2 + 1) % 5]
    elif kolumna_1 == kolumna_2:
        tekst = tablica[((wiersz_1 + 1) % 5) * 5 + kolumna_1]
        tekst += tablica[((wiersz_2 + 1) % 5) * 5 + kolumna_2]
    else:
        tekst = tablica[wiersz_1 * 5 + kolumna_2]
        tekst += tablica[wiersz_2 * 5 + kolumna_1]
    return tekst
```

Przykład 9

Przykład wywołania funkcji `znajdz_szyfr_pary()`:

```
# przykład wykonania dla zdefiniowanej tablicy
t = sekretna_tablica("szyfrogram")
print(znajdz_szyfr_pary(t, ("L","I")))
# wyjście
# PE
print(znajdz_szyfr_pary(t, ("N","U")))
# wyjście
# KW
print(znajdz_szyfr_pary(t, ("X","T")))
# wyjście
# TU
```



Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Możemy zauważyć, że w przypadku litery `X` zaszyfrowanym znakiem jest `T`.

Podobnie możemy zaprojektować funkcję **odszyfrowującą**, która będzie oparta na tabeli znaków.

Ponownie odczytujemy pozycję znaków w tablicy za pomocą funkcji `index()` oraz `divmod()`. Następnie odszyfrowujemy znaki.

```
def odszyfruj_pare(tablica: list, para_znakow: tuple):
    znak_1, znak_2 = para_znakow
    wiersz_1, kolumna_1 = divmod(tablica.index(znak_1), 5)
    wiersz_2, kolumna_2 = divmod(tablica.index(znak_2), 5)

    if wiersz_1 == wiersz_2:
        tekst = tablica[wiersz_1 * 5 + (kolumna_1 - 1) % 5]
        tekst += tablica[wiersz_2 * 5 + (kolumna_2 - 1) % 5]
    elif kolumna_1 == kolumna_2:
        tekst = tablica[((wiersz_1 - 1) % 5) * 5 + kolumna_1]
        tekst += tablica[((wiersz_2 - 1) % 5) * 5 + kolumna_2]
    else:
        tekst = tablica[wiersz_1 * 5 + kolumna_2]
        tekst += tablica[wiersz_2 * 5 + kolumna_1]
    return tekst
```

> **Przykład 10**

Przykład wywołania funkcji `odszyfruj_pare()`:

```
t = sekretna_tablica("szyfrogram")
print(odszyfruj_pare(t, ("P","E")))
# wyjście
# LI
print(odszyfruj_pare(t, ("K","W")))
# wyjście
# NU
print(odszyfruj_pare(t, ("T","U")))
#wyjście
#XT
```

Możemy zauważyć, że funkcja działa poprawnie także dla znaków brzegowych, jak na przykład znak T.

Mając powyższe elementy, możemy zapisać funkcję szyfrującą oraz odszyfrowującą. Musimy pamiętać, aby tekst do zaszyfrowania podać w postaci znaków występujących w tablicy szyfrującej.

```
def szyfruj_playfair(tekst_jawny: str, klucz: str):
    # tworzymy tablicę szyfrowania
    tablica = sekretna_tablica(klucz)
    # przetwarzamy tekst jawny na szyfrogram
    szyfrogram = ""
    for para in zwroc_pary(tekst_jawny):
        szyfrogram += znajdz_szyfr_pary(tablica, para)
    # zwracamy szyfrogram
    return szyfrogram
```

Przykład 11

```
# przykład wywołania
szyfrogram = szyfruj_playfair("Linuxtosystemoperacyjny", "klucz
print(szyfrogram)

# wyjście:
# KJMCPVIAFYVDNIVBASLRMOAT
```

Ponieważ proces szyfrowania dodał jeden znak na końcu naszego napisu, ostatni znak odszyfrowanej wiadomości będzie nadmiarowy. Funkcja deszyfrująca:

```
def odszyfruj_playfair(szyfrogram: str, klucz: str):
    # tworzymy tablicę szyfrowania
    tablica = sekretna_tablica(klucz)
    # przetwarzamy szyfrogram na tekst_jawny
    tekst_jawny = ""
    for para in zwroc_pary(szyfrogram):
        tekst_jawny += odszyfruj_pare(tablica, para)
    # zwracamy tekst jawny
    return tekst_jawny
```

Przykład 12

Przyład wywołania funkcji `odszyfruj_playfair()`:

```
# przykład wywołania
tekst_jawny = odszyfruj_playfair("KJMCPVIAFYVDNIVBASLRMOAT", "k
print(tekst_jawny)

# wyjście:
# LINUXTOSYSTEMOPERACYJNYX
```

Cały kod programu w języku Python wygląda następująco:

```
def sekretna_tablica(klucz: str):
    # dla tablicy 5 x 5 używamy alfabetu bez zanku "Q"
    alphabet = "ABCDEFGHIJKLMNOPRSTUVWXYZ"
    tablica = []

    # kopiujemy znaki, ignorujemy duplikaty
    for znak in klucz.upper():
        if znak not in tablica and znak in alphabet:
            tablica.append(znak)

    # wypełniamy pozostałe znaki w tablicy

    for znak in alphabet:
        if znak not in tablica:
```

```
            tablica.append(znak)
    return tablica

def zwroc_pary(tekst_jawny: str):
    # sprawdzamy długość tekstu
    if len(tekst_jawny) % 2 == 1:
        tekst_jawny += "x"
    tekst_jawny = tekst_jawny.upper()

    for i in range(0, len(tekst_jawny), 2):
        # zwracamy za pomocą yield kolejne pary znaków
        yield tekst_jawny[i:i+2]

def znajdz_szyfr_pary(tablica: list, para_znakow: tuple):
    znak_1, znak_2 = para_znakow
    wiersz_1, kolumna_1 = divmod(tablica.index(znak_1), 5)
    wiersz_2, kolumna_2 = divmod(tablica.index(znak_2), 5)

    if wiersz_1 == wiersz_2:
        tekst = tablica[wiersz_1 * 5 + (kolumna_1 + 1) % 5]
        tekst += tablica[wiersz_2 * 5 + (kolumna_2 + 1) % 5]
    elif kolumna_1 == kolumna_2:
        tekst = tablica[((wiersz_1 + 1) % 5) * 5 + kolumna_1]
        tekst += tablica[((wiersz_2 + 1) % 5) * 5 + kolumna_2]
    else:
        tekst = tablica[wiersz_1 * 5 + kolumna_2]
        tekst += tablica[wiersz_2 * 5 + kolumna_1]
    return tekst

def odszyfruj_pare(tablica: list, para_znakow: tuple):
    znak_1, znak_2 = para_znakow
    wiersz_1, kolumna_1 = divmod(tablica.index(znak_1), 5)
    wiersz_2, kolumna_2 = divmod(tablica.index(znak_2), 5)

    if wiersz_1 == wiersz_2:
        tekst = tablica[wiersz_1 * 5 + (kolumna_1 - 1) % 5]
        tekst += tablica[wiersz_2 * 5 + (kolumna_2 - 1) % 5]
    elif kolumna_1 == kolumna_2:
        tekst = tablica[((wiersz_1 - 1) % 5) * 5 + kolumna_1]
        tekst += tablica[((wiersz_2 - 1) % 5) * 5 + kolumna_2]
    else:
        tekst = tablica[wiersz_1 * 5 + kolumna_2]
```

```
57          tekst += tablica[wiersz_2 * 5 + kolumna_1]
58      return tekst
59
60  def szyfruj_playfair(tekst_jawny: str, klucz: str):
61      tablica = sekretna_tablica(klucz)
62      szyfrogram = ""
63      for para in zwroc_pary(tekst_jawny):
64          szyfrogram += znajdz_szyfr_pary(tablica, para)
65      return szyfrogram
66
67  def odszyfruj_playfair(szyfrogram: str, klucz: str):
68      tablica = sekretna_tablica(klucz)
69      tekst_jawny = ""
70      for para in zwroc_pary(szyfrogram):
71          tekst_jawny += odszyfruj_pare(tablica, para)
72      return tekst_jawny
73
74  # przykład wywołania
75  tekst_jawny = odszyfruj_playfair("KJMCPVIAFYVDNIVBASLRMOAT", "klu
76  print(tekst_jawny)
```

Dla zainteresowanych

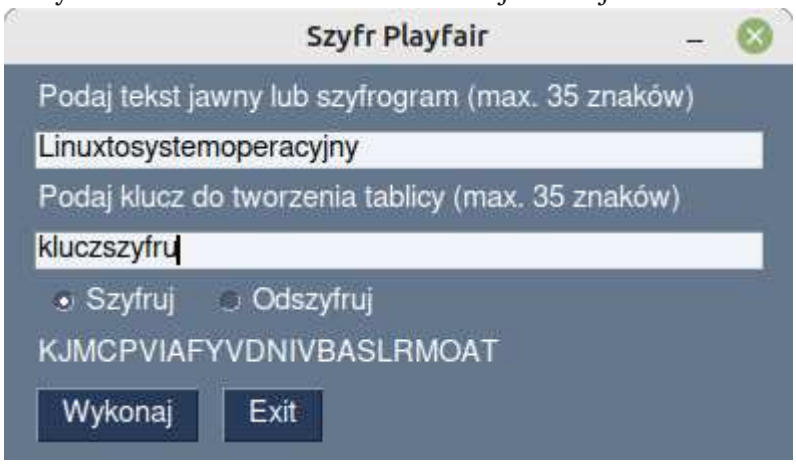
Możemy przygotować program z graficznym interfejsem dla szyfrowania lub odszyfrowywania. Użyjemy do tego celu biblioteki PySimpleGui.

```
1  def gui_szyfrowanie_playfair():
2      import PySimpleGUI as sg
3      uklad = [[sg.Text("Podaj tekst jawny lub szyfrogram (max. 3
4                [sg.Input(key="dane")],
5                [sg.Text("Podaj klucz do tworzenia tablicy (max. 3
6                [sg.Input(key="klucz")],
7                [sg.Radio("Szyfruj",1 , key="s", default=True),
8                 sg.Radio("Odszyfruj",1 , key="o")],
9                [sg.Text(" " * 40, size=(40,1), auto_size_text=Tru
10               [sg.Button('Wykonaj'), sg.Exit()]]
11     okno = sg.Window('Szyfr Playfair', uklad)
12
13     while True:
14         event, values = okno.read()
15         klucz_txt = values["klucz"]
```

```
16
17          if event == 'Exit' or event is None:
18              break
19          if event == 'Wykonaj':
20              if values['s']:
21                  wynik = szyfruj_playfair(values["dane"], klucz_
22                  okno["-OUT-"].update(wynik)
23              if values['o']:
24                  wynik = odszyfruj_playfair(values["dane"], kluc
25                  okno["-OUT-"].update(wynik)
26      okno.close()
27  # wywołanie funkcji
28  gui_szyfrowanie_playfair()
```

Efektem działania powyższej funkcji będzie okno z możliwością wyboru operacji.
Ważne!

Aby zobaczyć wynik operacji szyfrowania lub deszyfrowania, przed powyższym kodem trzeba umieścić wszystkie zdefiniowane wcześniej funkcje!



Słownik

divmod(*a, b*)

funkcja, która dla pary liczb całkowitych zwraca parę liczb będących wynikiem dzielenia całkowitego argumentów oraz resztą z takiego dzielenia;

przykład:

```
1  print(divmod(10, 5))
```

wynik:

```
(2, 0)
```

graficzny interfejs

sposób prezentacji informacji przez komputer oraz interakcji z użytkownikiem polegający na rysowaniu i obsługiwaniu widżetów

index()

funkcja zwracająca indeks `i` pierwszego wystąpienia danego elementu `x` w liście; jeżeli element nie występuje, zwracany jest błąd `ValueError: substring not found`, co oznacza, że danego elementu nie znaleziono;

przykład:

```
dni_tygodnia = ['poniedzialek', 'wtorek', 'sroda', 'czwartek',
print(dni_tygodnia.index('sroda'))
```

wynik:

```
2
```

PySimpleGUI

biblioteka do wyświetlania prostych okien dialogowych, niezależna od systemu operacyjnego; nie jest dostępna w standardowej instalacji języka Python - należy ją zainstalować, korzystając z mechanizmu `pip`

yield

wyrażenie służące – podobnie jak instrukcja `return` – do zwracania wyniku; tworzy tzw. generator, czyli funkcję, która po zwróceniu wyniku zapamiętuje stan zmiennych lokalnych i w kolejnych wywołaniach wykorzystuje zapamiętane wartości do zwracania kolejnych wyników

szyfrogram

zaszyfrowana wiadomość

Prezentacja multimedialna

Polecenie 1

Przeanalizuj proces szyfrowania metodą Playfair. Przeprowadź podobną analizę dla tekstu jawnego `HICSUNT` oraz klucza `LEONES`, nie używając przy tym komputera.

Polecenie 2

Poszukaj informacji na temat tego, ile czasu współczesne superkomputery potrzebują, by złamać szyfr Playfair.

1

Charles Wheatstone, wynalazca szyfru Playfair
Źródło: Wikimedia Commons, commons.wikimedia.org, dostęp 16.11.2022 r., domena publiczna.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

Do analizy użyjemy zmodyfikowanego kodu z sekcji „Przeczytaj”. Dodamy do niego komunikaty, które pomogą w śledzeniu działania algorytmu. Zacznijmy od przypomnienia

kolejnych kroków algorytmu szyfrowania szyfrem Playfair i funkcji, które je realizują.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

Pierwszym krokiem jest utworzenie 25-znakowej tabeli szyfrowania, zawierającej unikalne znaki słowa klucza oraz kolejne znaki alfabetu łacińskiego (poza znakiem Q), które nie wystąpiły w słowie kluczu.

Zadanie to realizuje funkcja `sekretna_tablica()`, w której dodajemy instrukcje wypisujące początkową tablicę (po dodaniu znaków słowa klucza) oraz kompletną tablicę (po dodaniu znaków alfabetu).

```
1  def sekretna_tablica(klucz: str):
2      """
3      Funkcja tworzy tablicę szyfrowania na podstawie słowa klucza i znaków alfabetu łacińskiego.
4      """
5
6      # używamy alfabetu bez znaku "Q", ponieważ tablica 5 x 5 będzie miała 25 znaków
7      alfabet = "ABCDEFGHIJKLMNOPRSTUVWXYZ"
8      tablica = []
9
10     # do tablicy dodajemy znaki z tekstu jawnego, pomijając te, które już są w tablicy
```

```
    for znak in klucz.upper():
        if znak not in tablica and znak in alfabet:
            tablica.append(znak)
    print(f"Początek tablicy : {tablica}")

    # do tablicy dodajemy znaki z alfabetu, pomijając te, które już są w tablicy
    for znak in alfabet:
        if znak not in tablica:
            tablica.append(znak)

    print(f"Cała tablica : {tablica}")
    return tablica

```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

3

Znaki tekstu jawnego szyfrowane będą parami. Potrzebujemy więc funkcji, która zwróci po dwa kolejne znaki na raz. Zadaniem funkcji jest również uzupełnienie tekstu jawnego znakiem x, jeżeli zawiera on nieparzystą liczbę znaków, a także przekształcenie go na wielkie litery.

```
def zwroc_pary(tekst_jawny):
    """
    Funkcja zwraca kolejne pary znaków z przekazanego
```

```
    tekstu jawnego.
4       """
5
6       # sprawdzamy, czy
    tekst jawny zawiera
    nieparzystą liczbę znaków
7       if len(tekst_jawny) %
    2 == 1:
8           print("Tekst jawny
    zawiera nieparzystą liczbę
    znaków – dodajemy 'x'.")
9           tekst_jawny += "x"
10
11      # zamieniamy znaki
    tekstu jawnego na duże
    litery
12      tekst_jawny =
    tekst_jawny.upper()
13
14      # za pomocą wyrażenia
    yield zwracamy kolejne
    pary znaków tekstu jawnego
15      for i in range(0,
    len(tekst_jawny), 2):
16          yield
    tekst_jawny[i:i+2]
17
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

Właściwe szyfrowanie realizuje funkcja `znajdz_szyfr_pary()`. Rozpoczynamy od określenia pozycji dwóch szyfrowanych znaków w 25-znakowej jednowymiarowej tablicy szyfrowania. Tablicę traktujemy jako dwuwymiarową – kolejne 5-znakowe ciągi są wierszami, a miejsce znaku szyfrowanego w takim ciągu to kolumna.

Numer wiersza i kolumny otrzymujemy po podzieleniu indeksu znaku w całej tablicy przez 5. Wynik całkowity oznacza wiersz, reszta z dzielenia to kolumna.

Następnie porównujemy wiersze i kolumny zajmowane przez parę znaków. W zależności od spełnionego warunku wyznaczamy zaszyfrowane znaki i zapisujemy w zmiennej, którą wypisujemy i zwracamy.

```
1  def
   znajdz_szyfr_pary(tablica:
   list, para_znakow: tuple):
2      """
3      Funkcja szyfruje
   przekazaną parę znaków
   tekstu jawnego:
4      - wyznacza pozycję,
   czyli wiersz i kolumnę,
   każdego znaku jawnego
5      - na podstawie
   warunków szyfrowania
   wyznacza pozycję znaków
   szyfrujących
6      - zwraca parę
   zaszyfrowanych znaków
7      """
8      znak_1, znak_2 =
   para_znakow
9      wiersz_1, kolumna_1 =
   divmod(tablica.index(znak_
   1), 5)
10     wiersz_2, kolumna_2 =
   divmod(tablica.index(znak_
   2), 5)
11     print(f"Pozycja znaku
   {znak_1} (wiersz/kolumna):
   {wiersz_1}/{kolumna_1}")
12     print(f"Pozycja znaku
   {znak_2} (wiersz/kolumna):
   {wiersz_2}/{kolumna_2}")
13
14     if wiersz_1 ==
   wiersz_2:
```

```
15          print("Znaki
znajdują się w tym samym
wierszu.")
16          tekst =
tablica[wiersz_1 * 5 +
(kolumna_1 + 1) % 5]
17          tekst +=
tablica[wiersz_2 * 5 +
(kolumna_2 + 1) % 5]
18      elif kolumna_1 ==
kolumna_2:
19          print("Znaki
znajdują się w tej samej
kolumnie.")
20          tekst =
tablica[((wiersz_1 + 1) %
5) * 5 + kolumna_1]
21          tekst +=
tablica[((wiersz_2 + 1) %
5) * 5 + kolumna_2]
22      else:
23          print("Znaki
znajdują się w różnych
wierszach i kolumnach.")
24          tekst =
tablica[wiersz_1 * 5 +
kolumna_2]
25          tekst +=
tablica[wiersz_2 * 5 +
kolumna_1]
26      print(f"Zaszyfrowana
para znaków: {tekst}.")
27      return tekst
28
```


Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

Wszystkie omówione funkcje umieszczamy w jednym pliku i dodajemy funkcję sterującą oraz

jej wywołanie. Funkcja `szyfruj_playfair()` pobiera jako argumenty tekst jawny i słowo klucz. Następnie tworzy tablicę szyfrowania, wywołując funkcję `sekretna_tablica()`. Dalej w pętli zaczyna odczytywanie kolejnych par znaków z tekstu jawnego, zwracanych przez funkcję `zwroc_pary()`. Każda para zostaje przekazana do funkcji `znajdz_szyfr_pary()`, która zwraca zaszyfrowane znaki – zostają one dopisane do zmiennej `szyfrogram`.

```
1  def szyfruj_playfair(tekst_jawny: str, klucz: str):
2      """
3      Funkcja szyfruje przekazany tekst jawny za pomocą słowa klucza.
4      """
5      # tworzymy tablicę szyfrowania
6      tablica = sekretna_tablica(klucz)
7      # przetwarzamy tekst jawny na szyfrogram
8      szyfrogram = ""
9      for para in zwroc_pary(tekst_jawny):
10
       print(f"Przetwarzamy parę znaków: {para}.")
11             szyfrogram += znajdz_szyfr_pary(tablica, para)
12
       print(f"Zaszyfrowany komunikat to: {szyfrogram}.")
13             print("-------------------------------------------")
```

```
    print(f"Koniec przetwarzania - szyfrogram to: {szyfrogram}.")
    return (szyfrogram)

szyfrogram = szyfruj_playfair("LINUXTOSYSTEM", "OPENSOURCE")
print(szyfrogram)

```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

Prześledzimy teraz przebieg algorytmu szyfrowania dla tekstu `LINUXTOSYSTEM` i klucza `OPENSOURCE`.

Po uruchomieniu programu zobaczymy wydruki początku i całej tablicy szyfrowania:

Początek tablicy zawiera tylko unikalne litery ze słowa klucza:

```
['O', 'P', 'N', 'S', 'U', 'R', 'C']
```

Cała 25-znakowa tablica zawiera dodatkowo nieobecne w niej wcześniej litery alfabetu.

```
['O', 'P', 'E', 'N', 'S',
 'U', 'R', 'C', 'A', 'B',
 'D', 'F', 'G', 'H', 'I',
 'J', 'K', 'L', 'M', 'T',
 'V', 'W', 'X', 'Y', 'Z']
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

Po informacji o uzupełnieniu tekstu jawnego, ze względu na nieparzystą liczbę znaków, zobaczymy komunikaty związane z szyfrowaniem kolejnych par znaków.

```
Przetwarzamy parę znaków:
LI.
Pozycja znaku L
(wiersz/kolumna): 3/2
Pozycja znaku I
(wiersz/kolumna): 2/4
Znaki znajdują się w
różnych wierszach i
kolumnach.
Zaszyfrowana para znaków:
TG.
Zaszyfrowany komunikat to:
TG.
--------------------------
--------------------------
--
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

```
Przetwarzamy parę znaków:
NU.
Pozycja znaku N
(wiersz/kolumna): 0/3
Pozycja znaku U
(wiersz/kolumna): 1/0
Znaki znajdują się w
różnych wierszach i
kolumnach.
Zaszyfrowana para znaków:
OA.
```

```
6 Zaszyfrowany komunikat to:
  TGOA.
7 --------------------------
  --------------------------
  --
```

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

```
1 Przetwarzamy parę znaków:
  XT.
2 Pozycja znaku X
  (wiersz/kolumna): 4/2
3 Pozycja znaku T
  (wiersz/kolumna): 3/4
4 Znaki znajdują się w
  różnych wierszach i
  kolumnach.
5 Zaszyfrowana para znaków:
  ZL.
6 Zaszyfrowany komunikat to:
  TGOAZL.
7 --------------------------
  --------------------------
  --
```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

```
1 Przetwarzamy parę znaków:
  OS.
2 Pozycja znaku O
  (wiersz/kolumna): 0/0
3 Pozycja znaku S
  (wiersz/kolumna): 0/4
```

```
Znaki znajdują się w tym samym wierszu.
Zaszyfrowana para znaków: PO.
Zaszyfrowany komunikat to: TGOAZLPO.
--------------------------------------------------
```

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

```
Przetwarzamy parę znaków: YS.
Pozycja znaku Y (wiersz/kolumna): 4/3
Pozycja znaku S (wiersz/kolumna): 0/4
Znaki znajdują się w różnych wierszach i kolumnach.
Zaszyfrowana para znaków: ZN.
Zaszyfrowany komunikat to: TGOAZLPOZN.
--------------------------------------------------
```

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

```
Przetwarzamy parę znaków: TE.
```

```
Pozycja znaku T (wiersz/kolumna): 3/4
Pozycja znaku E (wiersz/kolumna): 0/2
Znaki znajdują się w różnych wierszach i kolumnach.
Zaszyfrowana para znaków: LS.
Zaszyfrowany komunikat to: TGOAZLPOZNLS.
--------------------------------------------------
```

13

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

```
Przetwarzamy parę znaków: MX.
Pozycja znaku M (wiersz/kolumna): 3/3
Pozycja znaku X (wiersz/kolumna): 4/2
Znaki znajdują się w różnych wierszach i kolumnach.
Zaszyfrowana para znaków: LY.
Zaszyfrowany komunikat to: TGOAZLPOZNLSLY.
--------------------------------------------------
```

14

Baron Lyon Playfair, popularyzator szyfru Playfair
Źródło: Wikimedia Commons, commons.wikimedia.org, dostęp 16.11.2022 r., domena publiczna.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PLh9DwDqO

Po odczytaniu wszystkich par znaków tekstu jawnego, pętla for kończy swoje działanie. Całość szyfrogramu jest gotowa.

```
Koniec przetwarzania – szyfrogram to: TGOAZLPOZNLSLY.
```


Polecenie 3

Opracuj notatkę podsumowującą najważniejsze informacje przedstawione w prezentacji.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Uzupełnij podaną funkcję `odszyfruj`, tak aby zwracała tekst jawny dla zaszyfrowanego szyfrem Playfair ciągu znaków oraz tablicy szyfrującej zwróconej przez funkcję `przetwarzaj_klucz()`.

Specyfikacja problemu:

Dane:

- `wiadomosc` – ciąg o dowolnej liczbie znaków, składający się z wielkich liter alfabetu łacińskiego, bez litery Q
- `klucz` – klucz wykorzystywany do utworzenia tablicy szyfrującej

Wynik:

- `jawny` – odszyfrowane słowo; ciąg znaków składający się z wielkich liter alfabetu łacińskiego, bez litery Q

Działanie swojego programu przetestuj dla szyfrogramu `BRDOWJZTKWMP` oraz klucza `wskaznik`.

Twoje zadania

1. Uzupełnij podaną funkcję `odszyfruj`, tak aby zwracała tekst jawny dla zaszyfrowanego szyfrem Playfair ciągu znaków oraz tablicy szyfrującej zwróconej przez funkcję `przetwarzaj_klucz()`.

Ćwiczenie 2

Uzupełnij kod tak, aby podana funkcja zwracała tablicę szyfrującą utworzoną ze słowa bazowego zapisanego od końca. Działanie programu przetestuj dla słowa `slowo_bazowe = "playfair"`.

Specyfikacja problemu:

Dane:

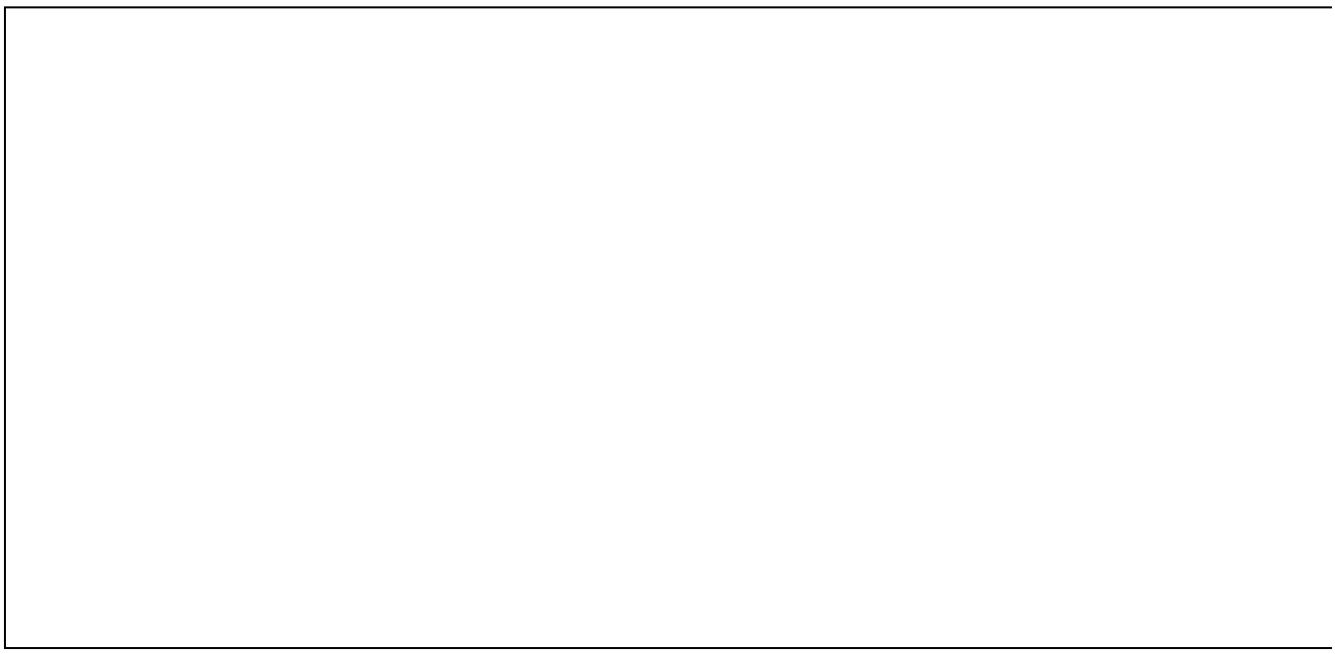
- `slowo_bazowe` – łańcuch znaków składający się z liter alfabetu łacińskiego bez litery Q; słowo, na podstawie którego tworzona jest tablica szyfrująca

Wynik:

- `tablica_szyfrujaca` – tablica szyfrująca utworzona ze znaków słowa bazowego `slowo_bazowe` (liter alfabetu łacińskiego bez litery Q) zapisanych od końca, wypisana w jednej linii poprzez zestawienie ze sobą wszystkich pięciu wierszy

Twoje zadania

1. Zmodyfikuj przedstawiony kod tak, aby podana funkcja zwracała tablicę szyfrującą utworzoną ze słowa bazowego zapisanego od końca.



Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Szyfr Playfair w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

V. Przestrzeganie prawa i zasad bezpieczeństwa. Respektowanie prywatności informacji i ochrony danych, praw własności intelektualnej, etykiety w komunikacji i norm współżycia społecznego, ocena zagrożeń związanych z technologią i ich uwzględnienie dla bezpieczeństwa swojego i innych.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

1) planuje kolejne kroki rozwiązywania problemu, z uwzględnieniem podstawowych etapów myślenia komputacyjnego (określenie problemu, definicja modeli i pojęć, znalezienie rozwiązania, zaprogramowanie i testowanie rozwiązania).

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres podstawowy. Uczeń:

1) projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);

2) do realizacji rozwiązań problemów prawidłowo dobiera środowiska informatyczne, aplikacje oraz zasoby, wykorzystuje również elementy robotyki;

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz zasadę działania szyfru Playfair.
- Przygotujesz funkcje realizujące szyfrowanie oraz deszyfrowanie metodą Playfair.
- Zaimplementujesz w języku Python program szyfrujący metodą Playfair.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych.

Formy pracy:

- praca indywidualna;
- praca w parach;

- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć**. Nauczyciel loguje się na platformie i udostępnia e-materiał: „Szyfr Playfair w języku Python". Uczniowie zapoznają się z multimedium w sekcji „Prezentacja multimedialna" oraz z sekcją „Przeczytaj".

Faza wstępna:

1. Nauczyciel prosi wybranych lub chętnych uczniów o wskazanie szyfrów, jakie znają. Prosi również o wskazanie, jakie widzą zastosowania dla szyfrów.
2. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie" i omawia cele do osiągnięcia w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel razem z uczniami omawia dziewięć przykładów przedstawionych w sekcji „Przeczytaj". Po każdym przykładzie uczniowie zapisują najważniejsze wnioski dotyczące tworzenia szyfrogramu za pomocą szyfru Playfair.
2. **Praca z multimedium.** Uczniowie zapoznają się z prezentacją.
3. **Ćwiczenie umiejętności.** Liga zadaniowa - uczniowie realizują indywidualnie ćwiczenia nr 1–2 z sekcji „Sprawdź się". Po ich wykonaniu nauczyciel omawia najlepsze rozwiązania problemów postawionych w zadaniach.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń.

Praca domowa:

1. Uczniowie wykonują Polecenie 1 z sekcji „Prezentacja multimedialna".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.
- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).

Wskazówki metodyczne:

- Przykład nr 10 z sekcji „Przeczytaj” może stanowić uzupełnienie wiedzy uczniów o zastosowaniu szyfru Playfair i biblioteki PySimpleGui.