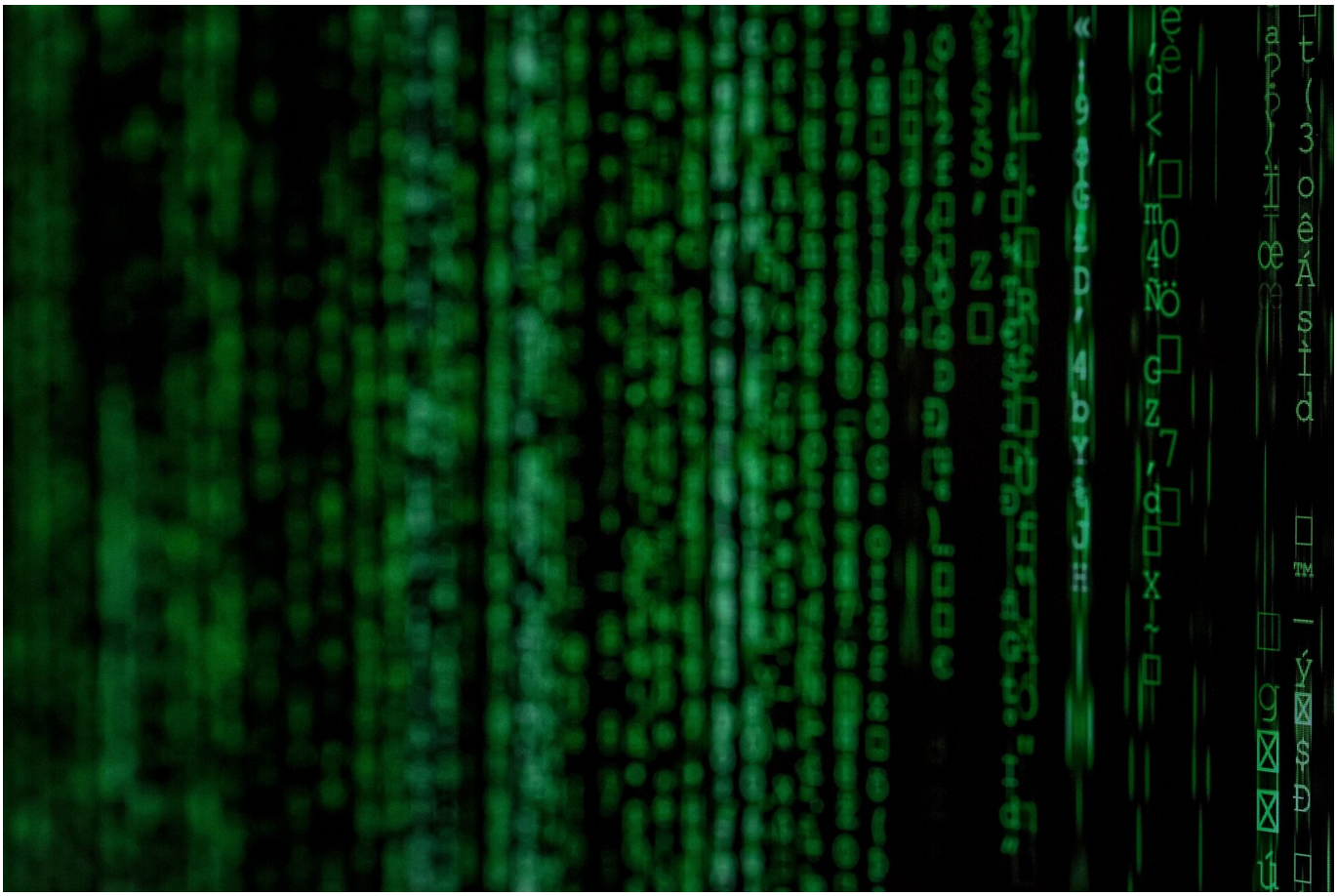


Wstęp do kryptografii w języku Python

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

Źródło: domena publiczna.

Ludzie od niepamiętnych czasów starają się chronić własne tajemnice oraz ważne informacje. Dlatego do komunikacji wykorzystują m.in. kryptografię czy steganografię.

Kryptografia to dziedzina nauki zajmująca się sposobami utajniania przekazywanych wiadomości. Steganografia również jest nauką, jednak dotyczy takiej komunikacji, w której ukryta ma być już sama obecność komunikatu. Innymi słowy, steganografia próbuje ukryć fakt prowadzenia komunikacji, a kryptografia – treści komunikatu.

Więcej teorii oraz ćwiczeń dotyczących tego zagadnienia znajdziesz w e-materiałach:

- Wstęp do kryptografii,
- Wstęp do kryptografii – zadania maturalne.

Ten e-materiał poświęcimy algorytmowi szyfrowania płotkowego. Zaimplementujemy go w programie napisanym 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:

- Wstęp do kryptografii w języku C++,
- Wstęp do kryptografii w języku Java.

Twoje cele

- Przeanalizujesz zasadę działania szyfru płotkowego.
- Przygotujesz funkcję realizującą szyfrowanie oraz deszyfrowanie przy użyciu szyfru płotkowego.
- Stworzysz program z graficznym interfejsem, wykorzystując bibliotekę `PySimpleGUI` do szyfrowania i odszyfrowywania wiadomości.

Przeczytaj

Szyfr płotkowy jest rodzajem szyfru przestawieniowego. Kluczem szyfrowania jest liczba naturalna, która określa, w ilu wierszach zapisujemy tekst jawny. Tworzenie szyfrogramu polega na zapisaniu liter tekstu jawnego zygzakiem, w dół i w górę, pomiędzy najwyższym i najniższym poziomem wymyślonego płotka o zadanej wysokości, która będzie jednocześnie tajnym kluczem. Następnie odczytujemy zapisane litery kolejno wierszami, aby zapisać je w jednej linii, tworząc szyfrogram wiadomości.

> **Przykład 1**
>
> Korzystając z szyfru płotkowego, zakodujemy przykładową wiadomość `NIHILNOVI`, przyjmując wartość klucza równą 3.
>
> **Przykładowe szyfrowanie:**
>
> Najpierw zapiszemy litery zygzakiem. Liczba wierszy jest równa wartości klucza, a więc w tym przypadku 3.
>
> ```
> N.......L.......I
> ..I...I...N...V
> ....H.......O
> ```
>
> Potem litery odczytujemy wierszami, rozpoczynając od pierwszego.
>
> - pierwszy wiersz zawiera litery: NLI,
> - drugi wiersz zawiera litery: IINV,
> - trzeci wiersz zawiera litery: HO.
>
> Po zakończeniu dla podanego tekstu otrzymamy szyfrogram:
>
> ```
> NLI IIN VHO
> ```
>
> ---
>
> Przygotujemy funkcję szyfrującą tekst metodą płotkową.
>
> **Specyfikacja problemu:**
>
> *Dane:*
>
> - `slowo` – łańcuch znaków; wiadomość do zaszyfrowania

- `klucz` – liczba naturalna > 1

Wynik:

- `szyfrogram` – łańcuch znaków; zaszyfrowana wiadomość

```
def szyfruj_slowo(slowo, klucz=5):
    # test typu danych wejściowych
    if type (slowo) is not str:
        return False

    if klucz < 2:
        return False

    # Krok 1. Usuwamy spacje i zmieniamy wszystkie litery na WI
    tekst_jawny = "".join([litera.upper() for litera in slowo i

    # Krok 2. Tworzymy listę list - płotek.
    liczba_znakow = len(tekst_jawny)
    plotek = [[" " for _ in range(klucz)] for _ in range(liczba

    # Krok 3. Rozmieszczamy kolejne znaki w listach
    # flaga kierunek = 1 - wypełniamy płotek w dół
    # flaga kierunek = -1 - wypełniamy płotek w górę

    kierunek = 1  # rozpoczynamy wypełnianie z góry na dół
    indeks = 0 # indeks kolejnego znaku jest wykorzystywany jak
    wiersz = 0  # nr wiersza, w którym aktualnie wstawiamy znak
    # gdy osiągniemy ostatni wiersz (zależy od wartości paramet
    # zmieniamy wartość zmiennej flaga
    while tekst_jawny:
        znak_tekstu = tekst_jawny[0]  # pierwszy znak
        plotek[indeks][wiersz] = znak_tekstu
        indeks += 1

        # teraz sprawdzamy warunek dla flagi
        if wiersz == klucz-1:  # w pythonie indeks zaczynamy li
            kierunek = -1
        elif wiersz == 0:
            kierunek = 1

        # teraz w zależności od flagi zwiększamy lub zmniejszam
```

```
        wiersz += kierunek

        # przepisujemy resztę tekstu
        tekst_jawny = tekst_jawny[1:]  # reszta jawnego tekstu,


    # Krok 4. odczytujemy zaszyfrowaną wiadomość
    szyfrogram = ""
    for wiersz in range(klucz):
        for wiersz_plotka in plotek:
            znak = wiersz_plotka[wiersz]
            if znak != " ": # jeśli mamy znak zamiast spacji
                szyfrogram += znak # dodajemy go do szyfrogramu

    # Zwracamy rezultat
    return szyfrogram
```

Wynik działania funkcji:

```
print(szyfruj_slowo("NIHILNOVI", 2))
print(szyfruj_slowo("NIHILNOVI", 3))
print(szyfruj_slowo("NIHILNOVI", 4))
print(szyfruj_slowo("NIHILNOVI", 5))
print(szyfruj_slowo("NIHILNOVI", 6))
print(szyfruj_slowo("NIHILNOVI", 1))
print(szyfruj_slowo("NIHILNOVI", -3))
print(szyfruj_slowo(123, 6))

# NHLOIIINV
# NLIIINVHO
# NOINVHLII
# NIIVHOINL
# NIHIIVLON
# False
# False
# False
```

Stwórzmy funkcję deszyfrującą zaszyfrowany tekst metodą płotkową.

Specyfikacja problemu:

Dane:

- `szyfrogram` – łańcuch znaków; wiadomość do zaszyfrowania
- `klucz` – liczba naturalna > 1

Wynik:

- `jawny_tekst` – łańcuch znaków; odszyfrowana wiadomość

```
def odszyfruj_slowo(szyfrogram, klucz):
    # test typu danych wejściowych
    if type (szyfrogram) is not str:
        return False

    if klucz < 2:
        return False

    # Odszyfrowywuje słowo na podstawie klucza

    # Przechowujemy informację o długości szyfrogramu
    # Będziemy wykorzystywać ją w pętli while ... przy uzupełni
    szyfrogram_len = len(szyfrogram)
    jawny_tekst = ""    # Odszyfrowane słowo
    # przygotowujemy listę list dla znaków szyfrogramu
    plotek = [["-" for _ in range(klucz)] for _ in range(szyfro


    # Wypełniamy listę * w miejscach liter
    # flaga kierunek = 1 - wypełniamy płotek w dół
    # flaga kierunek = -1 - wypełniamy płotek w górę

    kierunek = 1  # rozpoczynamy wypełnianie z góry na dół
    indeks = 0 # indeks kolejnego znaku jest wykorzystywany jak
    wiersz = 0  # nr wiersza, w którym aktualnie wstawiamy znak
    ile_znakow = szyfrogram_len
    # gdy osiągniemy ostatni wiersz (zależy od wartości paramet
    # zmieniamy wartość zmiennej flaga
    while ile_znakow:
        plotek[indeks][wiersz] = "*"
        indeks += 1

        # teraz sprawdzamy warunek dla flagi
```

```
        if wiersz == klucz-1:  # w pythonie indeks zaczynamy li
            kierunek = -1
        elif wiersz == 0:
            kierunek = 1

        # teraz w zależności od flagi zwiększamy lub zmniejszam
        wiersz += kierunek

        # zmniejszamy wielkość ilość znaków o 1
        ile_znakow -= 1

    # na kolejnych pozycjach gwiazdki wykładamy kolejne znaki

    while szyfrogram:
        for indeks in range(klucz):
            for wiersz in plotek:
                # ponieważ musimy iterować tyle razy, ile wynos
                # natrafimy kilka razy na wiersz, w którym znak
                # literą z szyfogramu; wtedy wiersz.index("*")
                # stosujemy konstrukcję try, aby nie przerwać s
                try:
                    if wiersz.index("*") == indeks: # miejsce "
                        wiersz[indeks] = szyfrogram[0] # wpisuj
                        szyfrogram = szyfrogram[1:] # pozostawi
                except:
                    pass
                    # przechodzimy dalej bez jakiegokolwiek kom
                    # aby nie zniekształcać obrazu

    # Uzupełniamy słowo analogicznie jak przy szyfrowaniu
    kierunek = 1  # rozpoczynamy wypełnianie z góry na dół
    indeks = 0 # indeks kolejnego znaku jest wykorzystywany jak
    wiersz = 0  # nr wiersza, w którym aktualnie wstawiamy znak
    ile_znakow = 0 # ile znakow odczytaliśmy, porównujemy do sz
    # gdy osiągniemy ostatni wiersz (zależy od wartości paramet
    # zmieniamy wartość zmiennej flaga
    while ile_znakow < szyfrogram_len:
        jawny_tekst += plotek[indeks][wiersz]
        indeks += 1

        # teraz sprawdzamy warunek dla flagi
        if wiersz == klucz-1:  # w pythonie indeks zaczynamy li
```

```
            kierunek = -1
        elif wiersz == 0:
            kierunek = 1

        # teraz w zależności od flagi zwiększamy lub zmniejszam
        wiersz += kierunek

        # zwiększamy ilość znaków zebranych
        ile_znakow += 1

    return jawny_tekst
```

Wynik działania funkcji:

```
print(odszyfruj_slowo("NHLOIIINV", 2))
print(odszyfruj_slowo("NLIIINVHO", 3))
print(odszyfruj_slowo("NOINVHLII", 4))
print(odszyfruj_slowo("NIIVHOINL", 5))
print(odszyfruj_slowo("NIHIIVLON", 6))
print(odszyfruj_slowo("NIHIIVLON", 1))
print(odszyfruj_slowo(124, 6))
print(odszyfruj_slowo("NIHIIVLON", -4))

# NIHILNOVI
# NIHILNOVI
# NIHILNOVI
# NIHILNOVI
# NIHILNOVI
# False
# False
# False
```

Ciekawostka

Błąd ValueError w języku Python występuje, gdy funkcja otrzymuje argument, który jest poza dopuszczalnym zakresem wartości lub jest nieodpowiedniego typu. Oto przykład:

```
number = int("abc")
# próba konwersji ciągu znaków na liczbę całkowitą,
# ale ciąg nie jest poprawnym numerem
```

W przykładzie funkcja `int()` oczekuje, że argument będzie ciągiem znaków, który reprezentuje liczbę całkowitą. Jednak w tym przypadku, ciąg „abc” nie może zostać przekonwertowany na liczbę całkowitą, co prowadzi do błędu ValueError.

Ten problem możemy rozwiązać, wykorzystując instrukcje `try` oraz `expect`:

```
try:
    number = int("abc")
except ValueError:
    print("Nie można przekonwertować ciągu znaków na liczbę cał
```

W poniższym przypadku sprawdzamy czy na liście występuje znak `"*"`. Używamy metody `lista.index("*")`, która zwraca indeks elementu listy, jeśli występuje. W innym przypadku zwraca błąd `ValueError`

```
lista_1 = [ "-", "-", "-", "*", "-", "-" ]
lista_2 = [ "-", "-", "-", "A", "-", "-" ]
lista_1.index("*")
# 3
lista_2.index("*")
# Traceback (most recent call last):
#   File "<pyshell#3>", line 1, in <module>
#     lista_2.index("*")
# ValueError: '*' is not in list
```

Rozwiązanie problemu:

```
lista_1 = [ "-", "-", "-", "*", "-", "-" ]
lista_2 = [ "-", "-", "-", "A", "-", "-" ]

try:
    index_1 = lista_1.index("*")
    print(f"Indeks gwiazdki w liście 1: {index_1}")
except ValueError:
    print("Gwiazdka nie jest na liście 1")

try:
    index_2 = lista_2.index("*")
    print(f"Indeks gwiazdki w liście 2: {index_2}")
```

```
except ValueError:
    print("Gwiazdka nie jest na liście 2")
```

Ważne!

Szyfr płotkowy jest łatwy do złamania metodą siłową, a więc sprawdzaniem każdej możliwej kombinacji. Wynika to z ograniczonej liczby możliwych do użycia kluczy.

Przykład 2

Dla dowolnego tekstu możemy przygotować prosty kod, który próbuje odszyfrować wiadomość metodą siłową, testując różne klucze.

Załóżmy, że zaszyfrowany tekst to `FAINTNOOIRPSOUCS`.

Ćwiczenie 1

Wskaż, jaki klucz został użyty do zaszyfrowania tekstu, jeśli tekst jawny to `FINISCORONATOPUS`.

- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5
- ◯ 6
- ◯ 7
- ◯ 8
- ◯ 9
- ◯ 10

Zapoznaj się z wywołaniem funkcji dla podanych w **Ćwiczeniu 1** wartości klucza:

```python
for klucz in range(2, 11):
    tekst = odszyfruj_slowo("FAINTNOOIRPSOUCS", klucz)
    print(f"Odszyfrowany tekst dla klucza {klucz} to: {tekst}")


# Odszyfrowany tekst dla klucza 2 to: FIARIPNSTONUOCOS
# Odszyfrowany tekst dla klucza 3 to: FTONAOUOIICRNPSS
# Odszyfrowany tekst dla klucza 4 to: FNIURTANPCSOIOOS
# Odszyfrowany tekst dla klucza 5 to: FIOPCSONATIOSURN
# Odszyfrowany tekst dla klucza 6 to: FINISCORONATOPUS
# Odszyfrowany tekst dla klucza 7 to: FINISUSCORONATOP
# Odszyfrowany tekst dla klucza 8 to: FINORSUSCOPIONAT
# Odszyfrowany tekst dla klucza 9 to: FANNORSUSCOPIOTI
# Odszyfrowany tekst dla klucza 10 to: FAINNORSUSCOPIOT
```

Przykład 3

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

> **Ważne!**
>
> Pamiętajmy, aby funkcje `szyfruj_slowo i odszyfruj_slowo` były zapisane w pliku o nazwie `funkcje_szyfrujace.py` i aby ten plik był w tym samym katalogu co tworzony plik z kodem interfejsu.

```python
import PySimpleGUI as sg
import sys

try:
    from funkcje_szyfrujace import *
except ModuleNotFoundError:
    print("Brak pliku: 'funkcje_szyfrujace.py' - popraw to.")
    # koniec programu z Error Code 2 (FileNotFound)
    sys.exit(2)

# jeśli funkcje zostały zaimportowane, działamy dalej...

uklad = [[sg.Text("Podaj tekst jawny lub szyfrogram (max. 35 zn
         [sg.Input(key="dane")],
         [sg.Radio("Szyfruj",1 , key="s", default=True),
          sg.Radio("Odszyfruj",1 , key="o"),
```

```
          sg.Text("Wartość klucza"), sg.Input(size=(3, 1), key=
         [sg.Text("Tekst wynikowy: ")],
         [sg.Text(" " * 40, size=(40,1), auto_size_text=True, k
         [sg.Button('Wykonaj'), sg.Exit()]]
okno = sg.Window('Szyfr płotkowy', uklad)

while True:
    event, values = okno.read()
    # wykonujemy próbę zmiany wartości na liczbę
    try:
        klucz = int(values["k"])
    # w przypadku niepowodzenia przyjmujemy 3 jako wartość kluc
    except:
        klucz = 3

    if event == 'Exit' or event is None:
        break
    if event == 'Wykonaj':
        if values['s']:
            wynik = szyfruj_slowo(values["dane"], klucz)
        if values['o']:
            wynik = odszyfruj_slowo(values["dane"], klucz)
        okno["-OUT-"].update(wynik)
okno.close()
```

Efektem działania takiego programu jest okno z możliwością wyboru operacji.

Już wiesz

Podsumujmy najważniejsze informacje:

- Szyfr płotkowy jest przykładem szyfru przestawieniowego.
- Szyfr płotkowy jest łatwy w implementacji.
- Atak siłowy jest efektywnym sposobem na złamanie takiego szyfru.

Słownik

iteracja

technika programowania, która polega na powtarzaniu tej samej operacji określoną liczbę razy lub do momentu, w którym zadany warunek zostanie spełniony

klucz

informacja wykorzystywana do szyfrowania i/lub deszyfrowania wiadomości

klucz publiczny

udostępniony publicznie klucz, wykorzystywany w procesie szyfrowania w szyfrach asymetrycznych

klucz prywatny

tajny klucz wykorzystywany w procesie deszyfrowania w szyfrach asymetrycznych; powinien być znany jedynie adresatowi zaszyfrowanej wiadomości

kryptografia

gałąź wiedzy o zapisywaniu informacji w sposób utrudniający lub całkowicie uniemożliwiający ich odczytanie

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`; (więcej informacji dostępnych jest w dokumentacji na stronie PySimpleGUI)

szyfrogram

zaszyfrowana wiadomość

szyfrowanie

przekształcanie tekstu jawnego w szyfrogram

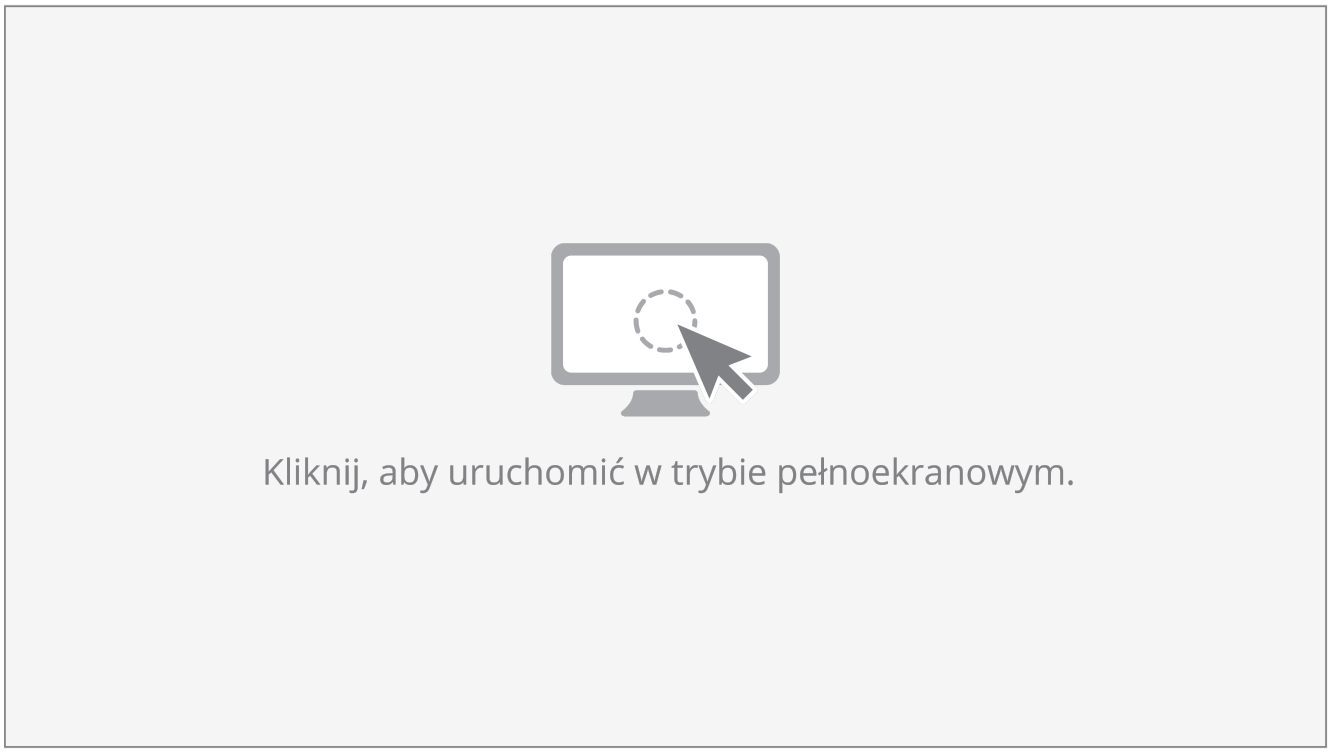
tablica ASCII

spis kodów znaków wykorzystywany w komputerach

Prezentacja multimedialna

Polecenie 1

Zapoznaj się z prezentacją. Prześledź działanie programu.



Polecenie 2

Zaszyfruj metodą płotkową tekst `Hic-sunt-leones`, używając klucza o wartości 3. Korzystaj z rozwiązania przedstawionego w prezentacji.

Specyfikacja problemu:

Dane:

- `slowo` – łańcuch znaków do zaszyfrowania
- `klucz` – liczba naturalna; klucz szyfru

Wynik:

- `szyfr` – zaszyfrowany łańcuch znaków

```
1
```

Polecenie 3

Opracuj notatkę podsumowującą najważniejsze informacje przedstawione w prezentacji.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Zdefiniuj funkcję `szyfruj(tekst)`, która jeśli długość łańcucha `tekst` będzie nieparzysta, doda na jego końcu znak „]”. Następnie funkcja ma podzielić łańcuch `tekst` na 2 równe części A i B i zwrócić nowy łańcuch zbudowany jako B+A, gdzie A powinno mieć odwróconą kolejność znaków.

Specyfikacja problemu:

Dane:

- `tekst` – łańcuch znaków do zaszyfrowania

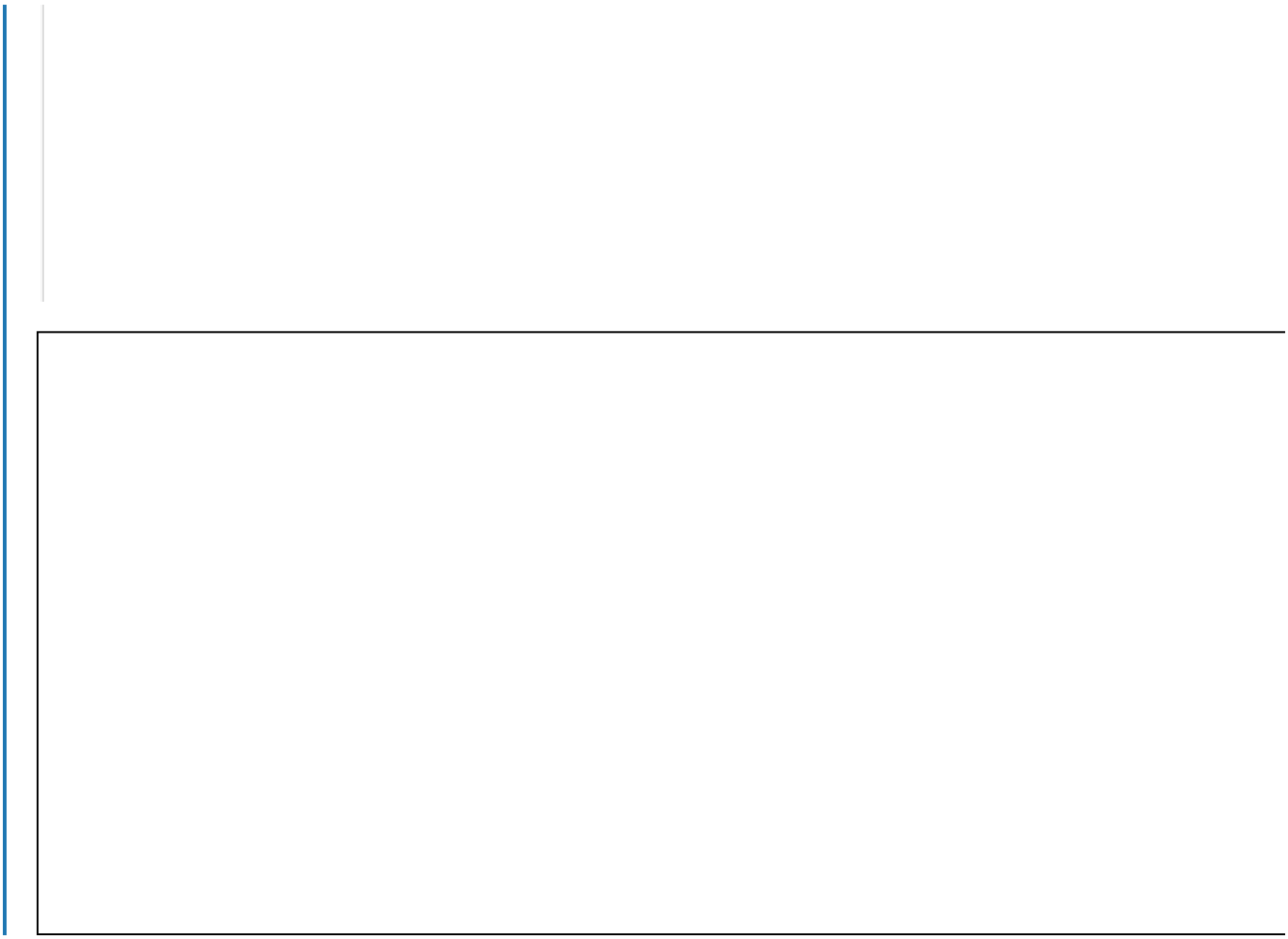
Wynik:

- `szyfr` – zaszyfrowany łańcuch znaków

Swój program przetestuj dla zmiennej `tekst` o wartości `Lupus in fabula`.

Twoje zadania

1. Program zwraca łańcuch znaków zaszyfrowany zgodnie z warunkami opisanymi w zadaniu.



Ćwiczenie 2

Zdefiniuj funkcję `szyfruj(tekst, klucz, odwrocona)`, która zrealizuje szyfrowanie łańcucha znaków `tekst` za pomocą szyfru płotkowego z kluczem `klucz` i zwróci zaszyfrowany łańcuch znaków. Wartość domyślna zmiennej `odwrocona` powinna być równa `False`, a gdy będzie wynosić `True`, do szyfrowania powinien posłużyć odwrócony łańcuch znaków `tekst`.

Specyfikacja problemu:

Dane:

- `tekst` – łańcuch znaków do zaszyfrowania
- `klucz` – liczba naturalna; klucz szyfru
- `odwrocona` – wartość logiczna

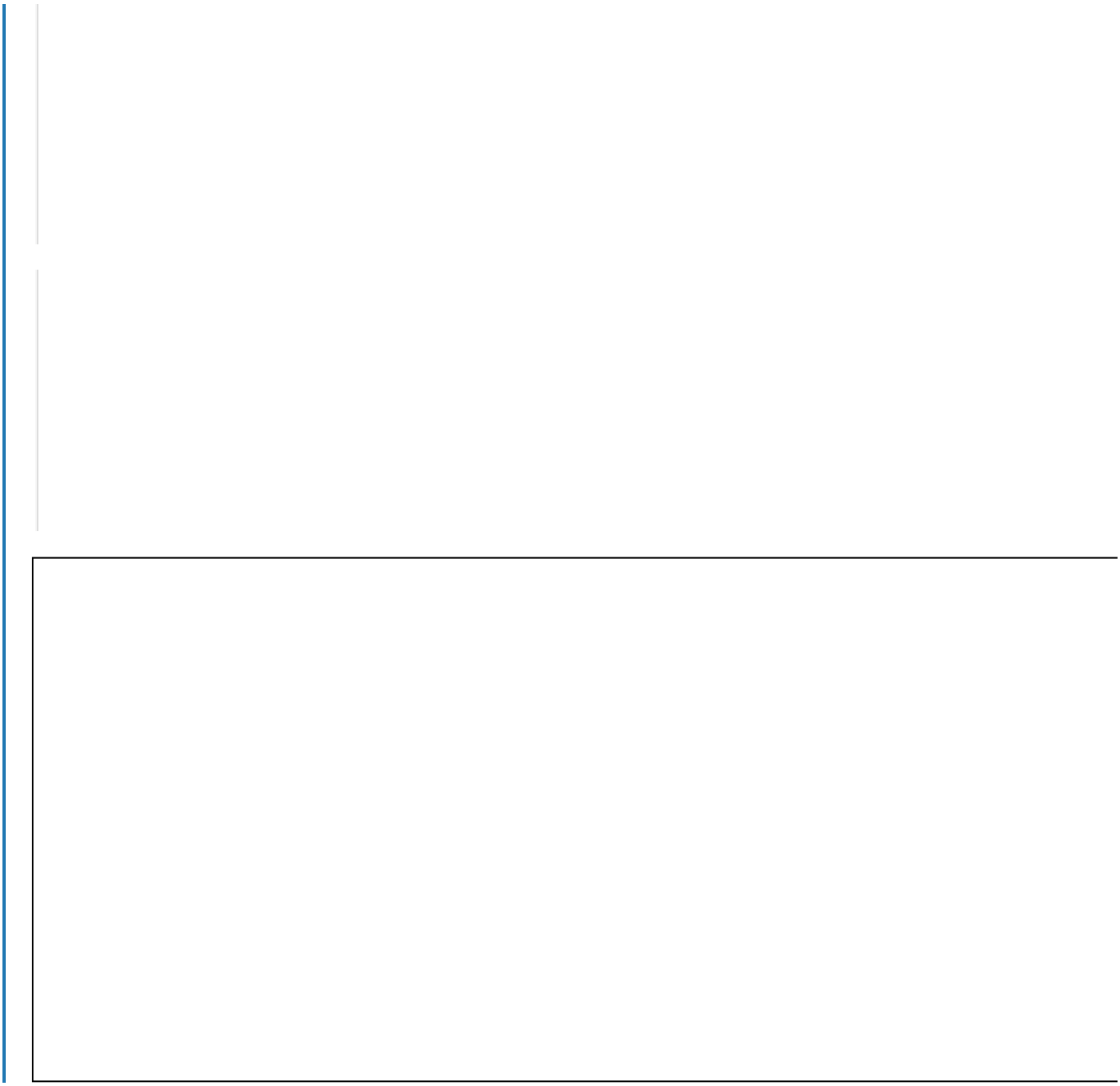
Wynik:

- `szyfr` – zaszyfrowany łańcuch znaków

Swój program przetestuj dla zmiennych `tekst` o wartości `Lupus in fabula`, `klucz` o wartości 3 i `odwrocona` o wartości `True`.

Twoje zadania

1. Program zwraca łańcuch znaków zaszyfrowany zgodnie z warunkami opisanymi w zadaniu.



Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Wstęp do kryptografii w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy

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).

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

b) na tekstach: porównywania tekstów, wyszukiwania wzorca w tekście metodą naiwną, szyfrowania tekstu metodą Cezara i przestawieniową,

4) porównuje działanie różnych algorytmów dla wybranego problemu, analizuje algorytmy na podstawie ich gotowych implementacji;

5) sprawdza poprawność działania algorytmów dla przykładowych danych.

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);

V. Przestrzeganie prawa i zasad bezpieczeństwa.

Zakres podstawowy. Uczeń:

3) stosuje dobre praktyki w zakresie ochrony informacji wrażliwych (np. hasła, pin), danych i bezpieczeństwa systemu operacyjnego, objaśnia rolę szyfrowania informacji;

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 płotkowego.
- Przygotujesz funkcję realizującą szyfrowanie oraz deszyfrowanie przy użyciu szyfru płotkowego.
- Stworzysz program z graficznym interfejsem, wykorzystując bibliotekę `PySimpleGUI` do szyfrowania i odszyfrowywania wiadomości.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

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

- ćwiczenia praktyczne.

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;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wstęp do kryptografii w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie" i omawia cele do osiągnięcia w trakcie lekcji.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytanie dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, opartego o programowanie.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj". W kolejnym kroku uczniowie powtarzają i testują na swoich komputerach programy omówione w tej sekcji.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna". Uczniowie wspólnie analizują proces szyfrowania metodą płotkową tekstu.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenie nr 1 z sekcji „Sprawdź się". Następnie chętne lub wybrane osoby prezentują swoje programy. Nauczyciel omawia ewentualne błędy.

Faza podsumowująca:

1. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku Python.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 2 z sekcji „Sprawdź się”.

Materiały pomocnicze:

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

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.