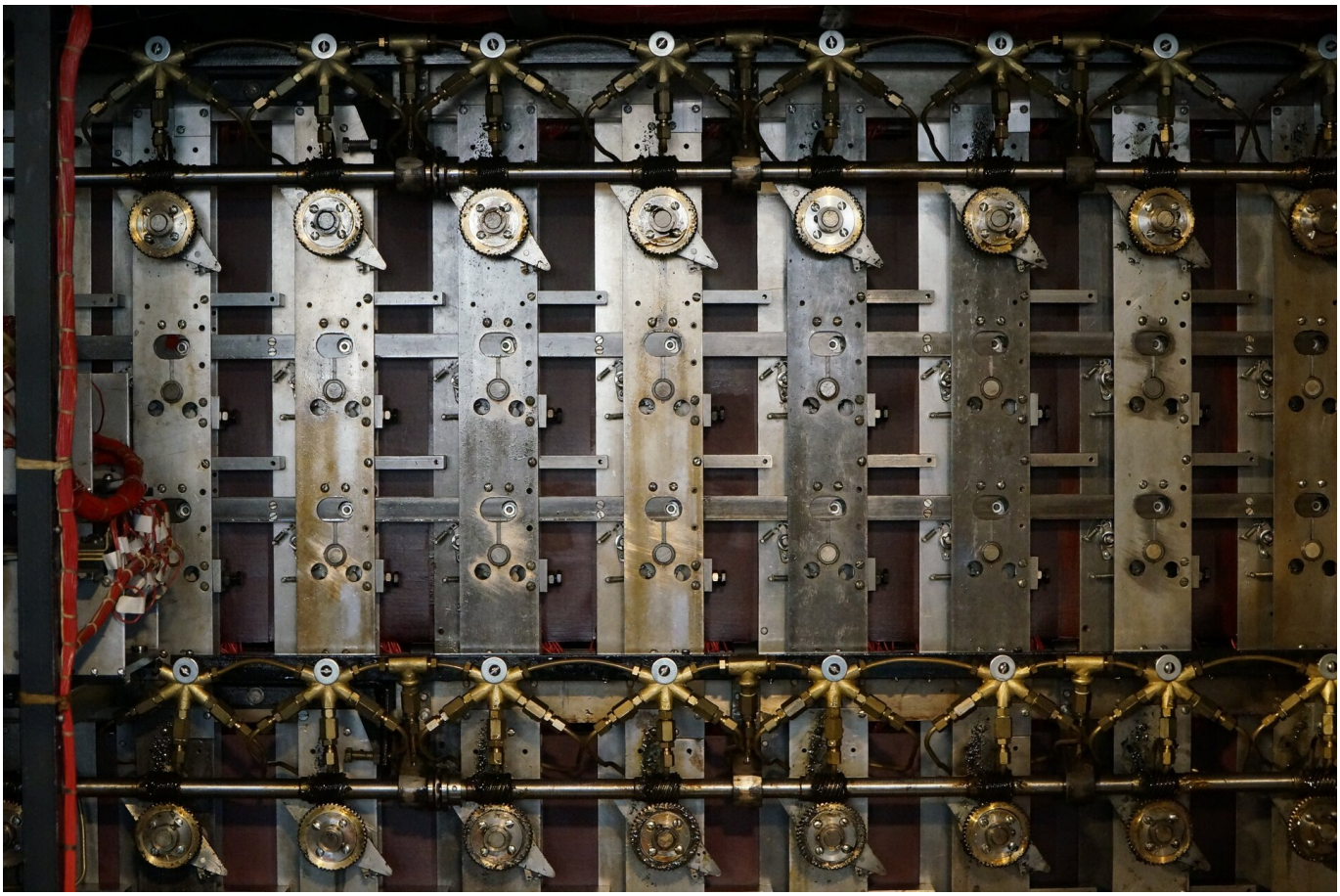

Szyfry symetryczne i asymetryczne w języku Python

Źródło: awsloley, domena publiczna.

Poznaliśmy już definicję szyfrów symetrycznych i asymetrycznych. Wiemy, czym się różnią, jak się dzielą oraz jakie jest ich zastosowanie. Kolejnym krokiem jest napisanie przykładowych programów szyfrujących.

Ten e-materiał poświęcony jest implementacji algorytmów realizujących szyfry symetryczne i asymetryczne w języku Python.

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

- Szyfry symetryczne i asymetryczne w języku C++,
- Szyfry symetryczne i asymetryczne w języku Java.

Więcej zadań? Sięgnij do: Szyfry symetryczne i asymetryczne – zadania maturalne.

> **Twoje cele**
>
> - Zaimplementujesz symetryczny algorytm ROT13 w języku Python.
> - Przeanalizujesz zasadę działania protokołu Diffiego-Hellmana, a następnie zaimplementujesz go w języku Python.
> - Rozwiążesz problemy, w których wykorzystasz znajomość szyfrów symetrycznych i asymetrycznych w języku Python.

Przeczytaj

Szyfry symetryczne w języku Python

Przykładem szyfru symetrycznego jest znany nam szyfr Cezara. Przeanalizujmy implementację jego szczególnego przypadku, znanego jako szyfr ROT13, w którym każdą literę przesuwamy o 13 pozycji.

> **Ważne!**
>
> Pamiętajmy, że w języku Python łańcuchy znaków są niezmienne – nie można ich modyfikować. Z tego względu w funkcji najpierw zamieniamy tekst wejściowy na listę, którą można modyfikować. Następnie zwracamy zmienną tekstową utworzoną z tej listy – korzystamy w tym celu z metody `.join()`.

Implementacja będzie znajdować się w funkcji `rot13`, przyjmującej i zwracającej zmienną typu tekstowego.

Przyjmijmy założenie, że tekst jawny zawiera małe litery alfabetu łacińskiego.

Zapiszmy nagłówek funkcji:

```
1 def rot13(tekst):
```

Zamieńmy teraz łańcuch znaków na listę znaków, a następnie utwórzmy pętlę `for`, która przejdzie po wszystkich elementach utworzonej tablicy:

```
1 def rot13(tekst):
2     wynik = list(tekst)
3 
4     for i in range(0,len(wynik)):
```

> **Ciekawostka**
>
> To nie jedyny sposób, by wykonać zamianę łańcucha znaków na listę znaków. Wykorzystać można również metodę `append()`.
>
> ```
> 1 def rot13(tekst):
> 2     wynik = []
> 3 
> ```

```
    for litera in tekst:
        wynik.append(litera)

    for i in range(0,len(wynik)):
        wynik[i] = chr((((ord(wynik[i]) - ord("a") + 13) % 26)

    wynik = "".join(wynik)

    print(wynik)

tekst = input("Podaj wiadomość do zaszyfrowana: ")

rot13(tekst)
```

W kolejnym kroku każdy ze znaków zamienimy poprzez zwiększenie jego kodu ASCII o 13. Jeżeli wykroczymy poza zakres liter, będziemy musieli powrócić do początku alfabetu. W tym celu zastosujemy operację reszty z dzielenia. Na końcu funkcji zwrócimy zmienną tekstową utworzoną z tablicy `wynik`.

```
def rot13(tekst):
    wynik = list(tekst)

    for i in range(0,len(wynik)):
        wynik[i] = chr((((ord(wynik[i]) - ord("a") + 13) % 26) +

    wynik = "".join(wynik)

    print(wynik)
```

Program prezentuje się następująco:

```
def rot13(tekst):
    wynik = list(tekst)

    for i in range(0,len(wynik)):
        wynik[i] = chr((((ord(wynik[i]) - ord("a") + 13) % 26) +

    wynik = "".join(wynik)

    print(wynik)
```

```
10
11 tekst = input("Podaj wiadomość do zaszyfrowana: ")
12
13 rot13(tekst)
```

Na szczególną uwagę w szyfrze ROT13 zasługuje fakt, że do odszyfrowania służy tu dokładnie ta sama funkcja, która zaszyfrowała tekst. Dzieje się tak dlatego, że w alfabecie łacińskim jest 26 znaków, zatem okresowość tego ciągu liter jest równa dwukrotności liczby 13. Przesuwając literę o 13 znaków dwa razy, otrzymujemy dokładnie tę samą literę. Zjawisko to występuje, ponieważ mamy tu do czynienia z szyfrem odwrotnościowym, czyli takim, w którym funkcja szyfrująca jest odwrotnością samej siebie.

Szyfry asymetryczne

Działanie szyfru asymetrycznego przeanalizujmy na podstawie protokołu Diffiego-Hellmana, który służy do tzw. wynegocjowania klucza publicznego między dwiema stronami. W przykładzie rozpatrzmy komunikację między klientem a serwerem.

1. Z góry ustalone są dwie liczby – liczba pierwsza p oraz podstawa g.
2. Klient losuje pewną liczbę całkowitą a i oblicza wartość $A = g^a \bmod p$. Wartość ta przesyłana jest do serwera.
3. Serwer analogicznie losuje liczbę b, po czym oblicza wartość $B = g^b \bmod p$. Następnie przesyła tę wartość do klienta.
4. Klient oblicza wartość $s = B^a \bmod p$, zaś serwer $s = A^b \bmod p$. Liczby te będą sobie równe.

Otrzymana wartość s jest sekretnym kluczem, który może być używany do szyfrowania symetrycznego za pomocą innego algorytmu. Wartości a oraz b są kluczami prywatnymi. Wszystkie pozostałe wartości w tym algorytmie mogą zostać udostępnione i będą miały niemal zerowy wpływ na siłę szyfru.

Prześledźmy teraz ten proces dla przykładowych danych:

1. Jako ustalone odgórnie wartości przyjmiemy liczbę pierwszą $p = 29$ oraz podstawę $g = 9$.
2. Klient wylosował wartość $a = 7$. Obliczona wartość A wynosi więc $A = g^a \bmod p = (29^7) \bmod 9 = 17249876309 \bmod 9 = 2$. Przesyłana jest ona do serwera.
3. Serwer wylosował liczbę $b = 3$. Obliczona wartość B wynosi więc $B = g^b \bmod p = (29^3) \bmod 9 = 24389 \bmod 9 = 8$. Przesyłana jest ona do klienta.

4. Obie strony obliczają wartość s. Dla klienta jest ona równa $s = B^a \bmod p = (B^7) \bmod 9 = 2097152 \bmod 9 = 8$. Dla serwera będzie równa $s = A^b \bmod p = (A^3) \bmod 9 = 8 \bmod 9 = 8$. Jak widać, obie otrzymane wartości są sobie równe.

Spójrzmy na implementację tego protokołu w języku Python. Będziemy kolejno zapisywać przedstawione wcześniej obliczenia:

```
import math

#elementy klucza publicznego
p = 29
g = 9

a = 7 #klucz prywatny klienta
print("Klucz prywatny klienta to ",a)
A = math.pow(g, a) % p; #element klucza publicznego klienta
print("A klienta to: ",A)

b = 3 #klucz prywatny serwera
print("Klucz prywatny serwera to ",b)
B = math.pow(g, b) % p; #element klucza publicznego serwera
print("B serwera to: ",B)

s1 = math.pow(B, a) % p; #klucz publiczny dla klienta
print("Klucz publiczny dla klienta to: ",s1)

s2 = math.pow(A, b) % p; #klucz publiczny dla serwera
print("Klucz publiczny na serwerze to: ",s2)
```

Szyfrowanie podczas przesyłania

Obecnie, korzystając z internetu, przesyłamy wiele informacji. Szczególnie więc zależy nam na tym, aby dane były bezpieczne. Takie zabezpieczenie udaje się osiągnąć poprzez szyfrowanie danych podczas ich przesyłania. Najczęściej stosuje się w tym celu protokół TLS, który może też służyć do zabezpieczenia takich protokołów jak FTP czy POP3. Służą one do wymiany plików i obsługi poczty e-mail. Sam protokół TLS to zbiór zasad opisujących bezpieczną komunikację – zawiera on instrukcje dotyczące zarówno szyfrów symetrycznych, jak i asymetrycznych.

Słownik

klucz prywatny

w szyfrze asymetrycznym sekretny klucz dostępny tylko i wyłącznie dla odbiorcy zaszyfrowanych wiadomości

klucz publiczny

w szyfrze asymetrycznym klucz, który może być powszechnie znany, bez wpływu na siłę szyfrowania

protokół TLS

protokół odpowiedzialny za zabezpieczanie informacji transportowanych w sieci; oparty na szyfrach asymetrycznych

szyfr odwrotnościowy

(ang. *reciprocal cipher*); szyfr, w którym funkcja szyfrująca, wywołana na zaszyfrowanym tekście, zwróci tekst oryginalny

Prezentacja multimedialna

Polecenie 1

Przeanalizuj prezentację. Przedstawimy w niej kod programu szyfrującego ciąg znaków za pomocą szyfru XOR, opartego na operacji logicznej alternatywy rozłącznej. Zapoznaj się z nim, a następnie uruchom go dla klucza `0xf`. Jak wygląda zaszyfrowany tekst? Jak myślisz, dlaczego otrzymano taki wynik?

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Napiszmy program, który zaszyfruje podany tekst za pomocą szyfru XOR.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

XOR to dwuargumentowa funkcja logiczna, która zwraca wartość prawda, wtedy gdy jeden i tylko jeden z argumentów jest prawdziwy. Dla obu wartości równych `True` lub `False` zwróci ona `False`. Jeżeli zaś wartości nie będą sobie równe, operacja zwróci wartość `True`.

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Zacznijmy od napisania pomocniczej funkcji, która będzie wyświetlała `tekst` w czytelny

sposób. Funkcja wypisze kody ASCII znaków na dwa sposoby – w postaci dziesiętnej i binarnej.

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

W kolejnym kroku wypiszemy kody ASCII w postaci dziesiętnej. Użyjemy do tego funkcji `ord()`.

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Aby ukończyć funkcję, należy wypisać kody ASCII w postaci binarnej. Użyjemy w tym celu funkcji `bin()`, która zwróci liczbę w postaci binarnej z dopiskiem `0b` na początku. Aby usunąć te dwa znaki, zastosujemy metodę tworzenia napisów w języku Python, za pomocą `[2:]`. Na końcu, aby liczba składała się z 8 cyfr, wywołamy funkcję `zfill(8)`, która dopisze odpowiednią liczbę zer na początku liczby.

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Następnie napiszmy funkcję szyfrującą. Będzie ona miała jeden parametr stanowiący tekst do zaszyfrowania.

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Przekształćmy zmienną z typu `string` na listę pojedynczych liter. Wywołajmy też funkcję `wyswietl`.

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

W pętli przekształćmy każdy znak na rezultat operacji XOR – na nim oraz na kluczu. W języku Python dokonujemy tego za pomocą znaku `^`. Kluczem będzie liczba heksadecymalna `0xf`.

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Ponownie wypisujemy wartości ASCII listy oraz zaszyfrowany tekst.

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Przykładowy kod demonstrujący działanie programu:

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PMJfPQEK2

Po uruchomieniu programu otrzymamy następujące wyniki:

Ważne!

W zaprezentowanym programie wykorzystaliśmy funkcję `zfill`. Dodaje ona zera na początku łańcucha znaków do momentu, kiedy nie osiągnie on wskazanej długości.

Przykład działania:

```
tekst = "zero"
x = tekst.zfill(5)
print(x)
```

Wynik działania programu:

```
0zero
```

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Zmodyfikuj podaną funkcję `rotN` tak, aby realizowała ona odwrotnościowy szyfr Cezara dla ciągu `x` złożonego z cyfr. Pamiętaj, że okres tego szyfru będzie wynosił 10. Działanie programu przetestuj dla `x = 12341234`.

Specyfikacja problemu:

Dane:

- x - ciąg znaków

Wynik:

- wartość logiczna

Twoje zadania

1. Funkcja `rotN` ma realizować szyfr odwrotnościowy dla ciągu złożonego z cyfr.

Ćwiczenie 2

Napisz program, który dla dwóch łańcuchów znaków (tekstu jawnego `jawny` oraz szyfrogramu `tajny`) znajdzie wartość, o jaką zostały przekształcone poszczególne znaki. Proces szyfrowania wygląda następująco:

```
1. W tekście jawnym zamień litery małe na wielkie, a wielkie na
2. Zapisz tekst jawny od końca.
3. Przekształć wszystkie znaki tekstu jawnego o naturalną liczb
```

Specyfikacja problemu:

Dane:

- `jawny` - tekst jawny, ciąg znaków
- `zaszyfrowany` - tekst zaszyfrowany, ciąg znaków

Wynik:

- liczba całkowita; wartość klucza

Twoje zadania

1. Program wypisuje jedną liczbę naturalną n będącą wartością, o jaką zostały przekształcone znaki tekstu.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Szyfry symetryczne i asymetryczne 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:

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,

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

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

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

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

f) metodę szyfrowania z kluczem publicznym i jej zastosowanie w podpisie elektronicznym,

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;

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

1) objaśnia rolę technik uwierzytelniania, kryptografii i podpisu elektronicznego w ochronie i dostępie do 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):

- Zaimplementujesz symetryczny algorytm ROT13 w języku Python.
- Przeanalizujesz zasadę działania protokołu Diffiego-Hellmana, a następnie zaimplementujesz go w języku Python.

- Rozwiążesz problemy, w których wykorzystasz znajomość szyfrów symetrycznych i asymetrycznych w języku Python.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne;
- odwrócona klasa.

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. Uczniowie przypominają sobie najważniejsze informacje dotyczące szyfrów symetrycznych i asymetrycznych.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Szyfry symetryczne i asymetryczne w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Następnie wspólnie z uczniami ustala kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

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". Następnie uczniowie analizują przedstawione w sekcji programy, powtarzają je i testują na swoich komputerach.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna". Uczniowie indywidualnie zapoznają się z treścią Polecenia 1 i realizują je, uprzednio analizując prezentację. Nauczyciel wyjaśnia ewentualne wątpliwości.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1 z sekcji „Sprawdź się", a następnie porównują swoje rozwiązania z kolegą lub koleżanką.
4. Chętne lub wybrane osoby przedstawiają swoje rozwiązanie ćwiczenia 1 z sekcji „Sprawdź się".

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Sprawdź się".
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 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:

- Nauczyciel może wykorzystać multimedium w sekcji „Prezentacja multimedialna" do pracy przed lekcją. Uczniowie zapoznają się z jego treścią i przygotowują do pracy na zajęciach w ten sposób, żeby móc samodzielnie rozwiązać zadania dołączone do e-materiału „Szyfry symetryczne i asymetryczne w języku Python".