Łańcuchy znaków w języku Python

Bibliografia:

- Źródło: Paulina Piotrowska, *ASCII ART*, 2009, dostępny w internecie: archive.org [dostęp 7.04.2021].

Łańcuchy znaków w języku Python

Źródło: Danielle MacInnes, domena publiczna.

> W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

Za każdym razem, gdy korzystamy z wyszukiwarki internetowej, w odpowiednie pole wpisujemy tekst. Wyszukiwarka udziela odpowiedzi na zapytanie również za pomocą tekstu. Można więc powiedzieć, że „mówimy tym samym językiem", a w konsekwencji – dzięki przetwarzaniu napisów – jesteśmy w stanie wykonywać operacje.

Skuteczne wyszukiwanie informacji nie jest jednak jedynym powodem, by poznać operacje na znakach. Dobra ich znajomość daje również możliwość szyfrowania informacji za pomocą różnych algorytmów.

W tym e-materiale dowiesz się, jak zbudowane są łańcuchy znaków (tzw. napisy), oraz poznasz sposoby operowania na nich w języku Python.

Implementacje łańcuchów znaków w pozostałych językach oprogramowania zostały omówione w e-materiałach:

- Łańcuchy znaków w języku C++,
- Łańcuchy znaków w języku Java.

Więcej zadań? Przejdź do e-materiału Łańcuchy znaków – zadania maturalne.

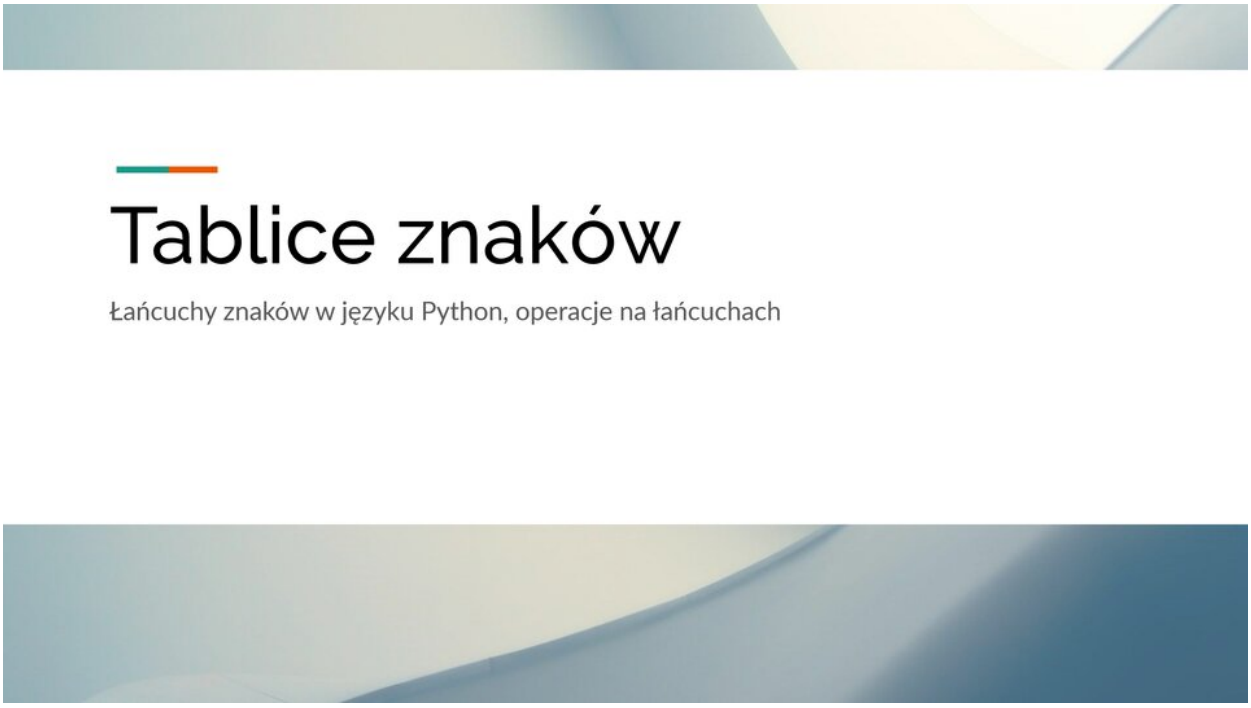
Twoje cele

- Przeanalizujesz kilka sposobów przechowywania obiektów tekstowych w języku Python.
- Prześledzisz, jakie operacje można wykonywać na ciągach znaków.
- Zinterpretujesz niektóre metody operujące na tekstach.
- Przeanalizujesz sposoby kodowania i dekodowania tekstu, wykorzystujące funkcje `chr() i ord()`.
- Użyjesz biblioteki `art` służącej do generowania tzw. `ASCII ART`.

Film samouczek

Polecenie 1

Przypomnij sobie, czym charakteryzują się łańcuchy znaków w języku Python. Następnie zapoznaj się z filmem prezentującym przykładowe operacje na łańcuchu znaków. Następnie wykonaj polecenie 1.



Film dostępny pod adresem /preview/resource/R1ZS2UDiy1YCn

Film nawiązujący do tablic znaków.

Polecenie 2

Dany jest łańcuch znaków `zdanie`. Napisz program, który:

1. utworzy nowy łańcuch składający się z łańcucha `zdanie` i łańcucha znaków `noweZdanie` dołączonego na końcu;

2. podzieli zdanie na pojedyncze wyrazy (jako separatora używając spacji), wynik podziału zapisze w liście, a następnie pobierze od użytkownika indeks słowa zapisanego w liście, wartość zmiennej n i wypisze co n-tą literę tego słowa;

3. korzystając z listy utworzonej w punkcie 2, pobierze od użytkownika indeks słowa zapisanego w liście i wypisze to słowo od końca.

Przetestuj swój program dla następujących danych:

- `zdanie = "Errare humanum est"`
- `noweZdanie = "sed in errare perseverare diabolicum"`
- `n = 2`

Specyfikacja problemu:

Dane:

- `zdanie` – łańcuch znaków
- `noweZdanie` – łańcuch znaków
- n – liczba naturalna

Wynik:

- Wypisane w osobnych liniach łańcuchy znaków będące wynikami operacji 1–3.

```
zdanie = "Errare humanum est"
noweZdanie = "sed in errare perseverare diabolicum"
n = 2

# Tu wpisz swój kod
```

```
1
```

Przeczytaj

W języku Python stosujemy dwie podstawowe funkcje pozwalające przeprowadzać operacje na kodach ASCII:

Funkcja	Przykład
`chr(liczba) -> znak`	`chr(65) -> 'A'`
`ord(znak) -> liczba`	`ord('P') -> 80`

> **Ważne!**
>
> Funkcja `chr()` przyjmuje jako argument liczbę całkowitą (kod ASCII), a zwraca odpowiadający jej znak.
>
> Funkcja `ord()` działa odwrotnie – jako argument przyjmuje znak, a zwraca odpowiadający mu kod ASCII.

Ciągi znaków w języku Python

W języku Python teksty są obiektami typu `string` (a dokładniej obiektami klasy `str`).

Oto przykłady definiowania i wyświetlania ciągów znaków:

```
1  napis = "Python"
2  print(type(napis))
3  # wypisze: <class 'str'>
4
5  napis = 'To jest napis 1'
6  print(napis)
7  # wypisze: To jest napis 1
8
9  napis = "A to inny sposób zapisu"
10 print(napis)
11 # wypisze: A to inny sposób zapisu
12
13 # Powyższe dwa zapisy są równoważne, natomiast
14 # wielolinijkowy napis możemy zapisać przy użyciu
15 # potrójnych cudzysłowów lub potrójnych apostrofów
16 napis = '''
17 A to jest napis,
```

```
który składa się z kilku linijek
'''
print(napis)

# wypisze: A to jest napis,
#          który składa się z kilku linijek

napis = "I znaki specjalne, np. tabulator \t lub nowa linijka \n
print(napis)
# wypisze: I znaki specjalne, np. tabulator      lub nowa linijka
#          tekstu.
```

Do sprawdzenia długości tekstu w języku Python służy funkcja `len()`:

```
dlugosc = len(napis)
print(dlugosc)
```

Obiekt typu `string` w języku Python jest niezmienną sekwencją. Do poszczególnych znaków możemy się odwołać za pomocą indeksu.

> **Przykład 1**
>
> Przeanalizujmy zdanie: „Jeśli rozwiązanie jest łatwe do wyjaśnienia, to możliwe, że jest ono dobrym rozwiązaniem" (sentencja to fragment ZEN języka Python tłumaczonego na język polski). Całe zdanie możemy potraktować jako ciąg pojedynczych znaków i odwoływać się do konkretnych elementów, podając ich indeksy (indeks pierwszego znaku wynosi 0).
>
> ```
> napis = "Jeśli rozwiązanie jest łatwe do wyjaśnienia, to możliw
>
> print(napis[0])
> # wypisze: J
>
> print(napis[15])
> # wypisze: i
>
> print(napis[18])
> # wypisze: j
> ```

Możemy również odwołać się do części napisu, używając wyrażeń indeksujących. Pozwala to tworzyć kod działający podobnie jak pętla, ale łatwiejszy do zapisania.

Oto zapis z wykorzystaniem pętli:

```
for i in range(18, 29):
    # end - ciąg znaków kończący dane wywołanie print(),
    #       domyślnie znak nowej linii '\n'.
    print(napis[i], end='')
# wypisze: jest łatwe
```

Identyczny efekt uzyskamy z użyciem wyrażenia indeksującego:

```
print(napis[18:29])
# wypisze: jest łatwe
```

Inne przykłady użycia wyrażeń indeksujących:

```
napis = "Ala ma kota"

# pominięcie pierwszego indeksu sprawia, że wycinanie zaczynamy
imie = napis[:3]
# wypisze: Ala
print(imie)

# pominięcie drugiego indeksu sprawia, że wycinanie kończymy na
zwierze = napis[7:]
# wypisze: kota
print(zwierze)

czasownik = napis[4:6]
# wypisze: ma
print(czasownik)
```

Ważne!

W ciągach znaków można odwoływać się do konkretnego elementu (podobnie jak w przypadku listy), ale nie da się zmieniać poszczególnych elementów – język Python

zgłosi w takim przypadku błąd:
`TypeError: 'str' object does not support item assignment`.

Oto przykład błędnego polecenia:

```
napis = "Python jest ciekawy"

napis[4] = "X"
# zgłosi błąd:
# Traceback (most recent call last):
#   File "<pyshell#7>", line 1, in <module>
#     napis[4] = "X"
# TypeError: 'str' object does not support item assignment
```

Dla zainteresowanych

Stosowanie wyrażeń indeksujących należy do tzw. pythonizmów. Warto je znać – pozwalają one uprościć tworzony program. Umiejętność działań na wycinkach (slicerach) przydaje się podczas operacji na tekstach. Wyrażenia umożliwiają zapisanie wspak podanego ciągu znaków. Przedstawiona poniżej instrukcja `print` oznacza: odczytuj napis od znaku ostatniego do pierwszego (czytaj cały tekst, zmniejszając indeksy, czyli od końca do początku).

```
napis = "Ala ma kota, a Python węża"
print(napis[::-1])
# wypisze: ażęw nohtyP a ,atok am alA
```

Działania na ciągach znaków: dodawanie i mnożenie

Oczywiście nie da się mnożyć tekstu przez tekst (przykładowo: `'Adam' * 'Linux'`). Możliwe jest natomiast mnożenie obiektów klasy `str` przez liczby typu `int` oraz dodawanie napisów do siebie.

Operacja	Dozwolona	Przykład
Dodawanie `str + str`	TAK	`'Adam ' + 'programista.'` -> `'Adam programista.'`
`Mnożenie str * str`	NIE	`'Adam' * 'Python'` -> `TypeError`
`Mnożenie str * int`	TAK	`4 * 'Python-'` -> `'Python-Python-Python-Python-'`

Przykład operacji dodawania:

```
napis_1 = 'Adam'
napis_2 = 'to programista'
napis_3 = 'Pythona'
napis = napis_1 + ' ' + napis_2 + ' ' + napis_3
print(napis)
# wypisze: Adam to programista Pythona
```

Operacja mnożenia:

```
napis = 4 * 'Python-'
print(napis)
# wypisze: Python-Python-Python-Python-
```

> **Polecenie 1**
>
> Przygotujmy funkcję, która będzie tworzyła prostokąt o podanych wymiarach, wypełniony znakami – przykładowo: `prostokat(x, y, znak)`.
>
> **Specyfikacja problemu:**
>
> *Dane:*
>
> - `x` – szerokość tworzonego prostokąta; liczba naturalna dodatnia
> - `y` – wysokość tworzonego prostokąta; liczba naturalna dodatnia
> - `znak` – litera, z której będzie tworzony prostokąt; znak
>
> *Wynik:*
>
> Program wypisuje stworzony ze znaków `znak` prostokąt o zadanych wymiarach.

> **Dla zainteresowanych**

W klasie `str` dostępna jest metoda `center()`. Pozwala ona osiągnąć interesujące efekty podczas pracy z tekstami. Przykładowo: polecenie `' + '.center(15,'$')` da wynik: `'$$$$$$ + $$$$$$'` – ustawi dany ciąg znaków na środku, tak aby łączna długość wynikowego ciągu znaków wynosiła 15, a puste miejsca uzupełni znakiem $.

Wszystkie metody klasy `str` możemy poznać, wydając polecenie `dir()`:

```
dir('tekst')
['__add__', '__class__', '__contains__', '__delattr__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__getitem__', '__getnewargs__',
 '__gt__', '__hash__', '__init__', '__init_subclass__',
 '__iter__', '__le__', '__len__', '__lt__', '__mod__',
 '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
 '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__',
 'capitalize', 'casefold', 'center', 'count', 'encode',
 'endswith', 'expandtabs', 'find', 'format', 'format_map',
 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
 'isidentifier', 'islower', 'isnumeric', 'isprintable',
 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
 'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
```

Proste kodowanie tekstu

Przejdźmy do kodowania. Możemy do tego celu wykorzystywać poznane właśnie elementy języka Python. Napiszemy dwie funkcje - pierwsza zamieni podany jej tekst w ciąg kodów ASCII, odpowiadających poszczególnym znakom, a druga wykona operację odwrotną, czyli zamieni ciąg kodów ASCII w tekst składający się ze znaków odpowiadających kolejnym kodom.

Przykładową implementację i efekty działania funkcji prezentuje następujący kod:

```
def zakoduj_napis(tekst):
    # tekst - zawiera aktualnie kodowany napis;
    # kod - przechowuje listę z kodami ASCII,
    #       odpowiadającą aktualnie kodowanemu tekstowi.
```

```
    kod = []
    for litera in tekst:
        kod.append(ord(litera))

    return kod

def dekoduj_napis(kodowany):
    # kodowany - lista z kodami ASCII
    #            znaków zakodowanego słowa;
    # napis - zawiera odkodowane znaki odpowiadające kolejnym
    #         kodom ASCII, z prefiksem 'Odkodowany: '.
    napis = 'Odkodowany: '
    for znak in kodowany:
        napis += chr(znak)

    return napis


zakodowany = zakoduj_napis('Adam -> programista')
print(zakodowany)
# wypisze: [65, 100, 97, 109, 32, 45, 62, 32, 112, 114, 111,
# 103, 114, 97, 109, 105, 115, 116, 97]

tekst = dekoduj_napis(zakodowany)
print(tekst)
# wypisze: Odkodowany: Adam -> programista
```

Polecenie 2

Napisz program odkodowujący tajny tekst zapisany za pomocą kodów ASCII. Dzięki temu odczytasz ciekawostkę (m.in. na temat oprogramowania FLOSS).

Przetestuj działanie programu dla tablicy:

```
tajny = [76, 105, 110, 117, 120, 32, 116, 111, 32, 99, 105,
         101, 107, 97, 119, 121, 32, 115, 121, 115, 116,
         101, 109, 32, 111, 112, 101, 114, 97, 99, 121,
         106, 110, 121, 32, 45, 32, 70, 76, 79, 83, 83]
```

Specyfikacja problemu:

Dane:

- `tajny` – wiadomość zakodowana za pomocą kodów ASCII; tablica liczb naturalnych

Wynik:

Program wypisuje odkodowaną wiadomość.

Operacje na dłuższych tekstach

Wykonajmy kilka operacji na fragmencie dokumentu zawierającego wiele zdań. Jako przykład posłuży nam zbiór felietonów Tadeusza Boya-Żeleńskiego pt. *Dziewice konsystorskie* (dostępny w domenie publicznej).

Oto wybrany fragment:

```
Co innego jest propagować coś, a co innego widzieć, że coś jest f
Mówić o tym - w potrzebie nawet krzyczeć - jest obowiązkiem pisar
```

W kolejnych e-materiałach dowiesz się, jak zapisać i wczytać tekst z pliku – teraz natomiast zapiszemy cały tekst w jednej zmiennej, z użyciem notacji pozwalającej na definiowanie napisów wielolinijkowych:

```
```

```
dane = """Co innego jest propagować coś, a co innego widzieć, że
Mówić o tym - w potrzebie nawet krzyczeć - jest obowiązkiem pisar
```

Ujęcie tekstu w znaki trzech cudzysłowów (lub trzech apostrofów) powoduje, że wszystkie znaki – włącznie ze znakami nowej linii – zostają zapisane do zmiennej `dane`. Możemy podzielić tekst na listę za pomocą metody `split()`. Konwertujemy wtedy tekst na listę, której elementami są łańcuchy znaków zawierające poszczególne linie.

```
lista_z_danymi = dane.split("\n")

for i in range(len(lista_z_danymi)):
    print(lista_z_danymi[i])
```

Funkcje `count()` i `find()`

W rozwiązaniu kolejnego problemu użyjemy dwóch ważnych funkcji: `count()` i `find()`.

Pierwsza z nich: `count()` służy do zliczania wystąpień danego wzorca w tekście. Jeżeli wzorzec nie występuje, zwraca ona wartość `0`.

Funkcja `find()` wyszukuje wzorzec w tekście i zwraca pozycję jego pierwszego wystąpienia. Funkcja zwraca wartość `-1`, jeżeli wzorzec nie występuje w tekście.

> **Polecenie 3**
>
> Znajdźmy odpowiedzi na następujące pytania:
>
> - **Z ilu zdań składa się tekst?** Dowiemy się tego, licząc kropki, wykrzykniki i znaki zapytania; wykorzystamy metodę `count()`.
> - **Ile spacji zawiera tekst?** Aby to sprawdzić, ponownie użyjemy metody `count()`.
> - **Ile słów „się” znajdziemy w tekście?** Tu znów sięgniemy po metodę `count()`.
> - **Na której pozycji zapisano pierwsze słowo „się”?** By się tego dowiedzieć, skorzystamy z metody `find()`.

> **Polecenie 4**
>
> Przykładowy tekst należy przekształcić w:
>
> - ciąg złożony z co piątego znaku, zaczynając od pierwszego znaku w tekście;
> - tekst, w którym słowa „się” zostaną zamienione na „XXX”, a „kiedy” na „ZZZZ”.

ASCII ART – sztuka zapomniana

Operując na ciągach znaków, można tworzyć oryginalne napisy. Służy do tego biblioteka ART, która jest projektem typu *Open Source*. Kod biblioteki znajdziemy w serwisie GitHub.

Funkcja `text2art` pozwala na takie przetworzenie łańcucha znaków, że zostaje on wyświetlony w formie graficznej za pomocą innego łańcucha znaków.

Funkcja `art` natomiast wyświetla graficzną reprezentację łańcucha znaków. Oto przykłady zastosowania:

```
from art import *

print(text2art("art", "block"))
# wypisze:
#  .----------------.  .----------------.  .----------------.
# | .--------------. || .--------------. || .--------------. |
# | |      __      | || |  _______     | || |  _________   | |
# | |     /  \     | || | |_   __ \    | || | |  _   _  |  | |
# | |    / /\ \    | || |   | |__) |   | || | |_/ | | \_|  | |
# | |   / ____ \   | || |   |  __ /    | || |     | |      | |
# | | _/ /    \ \_ | || |  _| |  \ \_  | || |    _| |_     | |
# | ||____|  |____|| || | |____| |___| | || |   |_____|    | |
# | |              | || |              | || |              | |
# | '--------------' || '--------------' || '--------------' |
#  '----------------'  '----------------'  '----------------'

print(art("men"))
# wypisze:
# ب_ب

print(art("woman"))
```

```
# wypisze:
# ▒⚗_⚗▒

print(art("car"))
# wypisze:
#  `o##o>
```

Dla zainteresowanych

W roku 2009 Paulina Piotrowska, studentka Akademii Sztuk Pięknych w Gdańsku, napisała pracę magisterską zatytułowaną „ASCII ART" (nr albumu 4836).

We wstępie czytamy:

> **Paulina Piotrowska**
>
> ***ASCII ART***
>
> ASCII Art, według mnie pierwsza i jedyna najbardziej skomputeryzowana forma przedstawienia grafiki w formie cyfrowej. Tworzona za pomocą pierwotnego narzędzia, jakiego używa współczesny grafik komputerowy – klawiatury. Dodatkowo tworzona w najbardziej podstawowym programie, którego używają programiści – edytorze tekstu.
>
> Źródło: Paulina Piotrowska, *ASCII ART*, 2009, dostępny w internecie: archive.org [dostęp 7.04.2021].

Już wiesz

Podsumujmy najważniejsze elementy tego e-materiału:

- Ciągi znaków w języku Python są obiektami klasy `str`, a składają się ze znaków ASCII.
- Ciągi znaków są niezmiennymi sekwencjami.
- Ciągi znaków możemy przetwarzać, stosując metody takie jak `split()`, `count()` czy `find()`.

Słownik

ART

moduł służący do graficznego przedstawienia napisu złożonego ze znaków ASCII; należy go zainstalować, korzystając z komendy `pip install art`

ASCII

(ang. *American Standard Code for Information Interchange*) system kodowania znaków, pierwotnie oparty na 7 bitach (współcześnie rozszerzony)

FLOSS

(ang. *Free/Libre and Open Source Software* – wolne oprogramowanie i otwarty kod źródłowy) programy, które są dostępne dla każdego zgodnie z zasadami przedstawionymi na stronie projektu GNU

niezmienna

(ang. *immutable*) sekwencja, której pojedynczych elementów nie da się zmodyfikować; w celu dokonania zmiany konieczne jest zastąpienie całego obiektu nowym (niezbędna jest wymiana całego ciągu, aby zastąpić w nim pojedynczy znak)

pythonizm

specyficzna dla języka Python konstrukcja programistyczna, pozwalająca na tworzenie krótkiego kodu

wyrażenie indeksujące

(ang. *slice*) zawarte w nawiasach kwadratowych wyrażenie, które pozwala określić wybraną część sekwencji; może mieć ono postać `[a:b]` lub `[a]` lub podobną; nazywane także `wycinkiem`

ZEN języka Python

manifest *ZEN of Python by Tim Peters* – dokument składający się z 19 punktów, które zostały napisane i opublikowane na liście dyskusyjnej języka Python w 1999 roku; treść manifestu można przeczytać, wydając polecenie `import this` w środowisku IDLE

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Ćwiczenie 2

Napisz program, który podzieli podany napis `dane` na listę napisów (znakiem podziału powinna być wartość zapisana w zmiennej `separator`), a następnie wydrukuje na standardowe wyjście otrzymaną listę, każdy element w osobnej linii.

Przetestuj swój program dla następujących danych:

- `dane = "1,Jan,Kowalski,26,Warszawa"`
- `separator = ","`

Specyfikacja problemu:

Dane:

- `dane` – napis, który ma zostać podzielony; łańcuch znaków
- `separator` – separator; znak podziału

Wynik:

- Program wypisuje w kolejnych liniach elementy wchodzące w skład listy napisów.

Ćwiczenie 3

Napisz program, który policzy, ile razy litera `litera` pojawia się w słowie `slowo`.

Przetestuj swój program dla następujących danych:

- `slowo = "lokomotywa"`
- `litera = "o"`

Specyfikacja problemu:

Dane:

- `slowo` – badane słowo; łańcuch znaków
- `litera` – szukany znak

Wynik:

- Program wypisuje liczbę wystąpień litery `litera` w słowie `slowo`.

Ćwiczenie 4

Napisz program, który odwraca kolejność liter w wyrazie `wyraz` i wyświetla na ekranie słowo zapisane na wspak.

Przetestuj swój program dla następujących danych:

- `wyraz = "informatyka"`

Specyfikacja problemu:

Dane:

- `wyraz` – napis, w którym należy odwrócić kolejność liter; łańcuch znaków

Wynik:

- Program wyświetla słowo `wyraz` zapisane od końca.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Łańcuchy znaków w języku Python

Grupa docelowa:

Liceum ogólnokształcące i technikum, 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.

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,

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

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury 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);

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 kilka sposobów przechowywania obiektów tekstowych w języku Python.
- Prześledzisz, jakie operacje można wykonywać na ciągach znaków.
- Zinterpretujesz niektóre metody operujące na tekstach.
- Przeanalizujesz sposoby kodowania i dekodowania tekstu, wykorzystujące funkcje `chr() i ord()`.
- Użyjesz biblioteki `art` służącej do generowania tzw. `ASCII ART`.

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

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Uczniowie powtarzają wiadomości dotyczące łańcuchów znaków oraz operacji na nich w języku Python.

Faza wstępna:

1. Przedstawienie tematu zajęć oraz wspólne z uczniami ustalenie kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie zapoznają się z filmem. Następnie indywidualnie opracowują rozwiązanie polecenia 2.
2. **Praca z tekstem.** Uczniowie zapoznają się z treściami zamieszczonymi w sekcji „Przeczytaj". Przygotowują notatkę podsumowującą najważniejsze informacje. W razie potrzeby nauczyciel odpowiada na pytania.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenie 1. Nauczyciel sprawdza poprawność wykonania.
4. W kolejnym kroku uczniowie pracując w parach, wykonują ćwiczenia 2 i 3. Po zakończeniu zadania porównują swoje rozwiązania z inną grupą.

Faza podsumowująca:

1. Wybrany uczeń podsumowuje zajęcia z programowania w Pythonie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 4 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 na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.