# Palindromy w języku Python


- Wprowadzenie
- Przeczytaj
- Animacja
- Sprawdź się
- Dla nauczyciela


Źródło: Wokandapix, domena publiczna.

Od niepamiętnych czasów ludzie, szukając rozrywki, bawili się również słowami. Zapewne w ten właśnie sposób powstały palindromy (z gr. *palindromeo* – „biec z powrotem"). Być może miały one także znaczenie magiczne... Podstawowe informacje na ten temat znajdziesz w e-materiale Palindromy.

My jednak zajmiemy się palindromami z zupełnie innego powodu: operując na ciągach znaków, nabędziemy większej wprawy w posługiwaniu się językiem Python.

Implementację programu sprawdzającego, czy dane słowo jest palindromem, w pozostałych językach programowania znajdziesz w e-materiałach:

- Palindromy w języku C++,
- Palindromy w języku Java.

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

> **Twoje cele**
>
> - Zastosujesz algorytm sprawdzający, czy dany ciąg znaków jest palindromem, wykorzystując pętlę `for`.
> - Użyjesz wyrażeń indeksujących do sprawdzenia, czy ciąg znaków jest palindromem.
> - Przeanalizujesz nowy sposób formatowania napisów – `f-string`.

# Przeczytaj

**Przykład 1**

Zdefiniujemy funkcję sprawdzającą, czy podane wyrażenie jest palindromem.

W tym celu wykorzystamy pętlę `for` oraz instrukcję warunkową `if ... else`.

Przyjmujemy następującą strategię: jeśli wyraz jest palindromem, to jego pierwsza litera będzie taka sama jak ostatnia, druga będzie taka sama jak przedostatnia itd. Dopóki nie dojdziemy do połowy wyrazu. Dalsze porównywanie nie ma sensu, ponieważ będziemy porównywać te same pary znaków.

```
def palindrom_petla(tekst):
    dlugosc = len(tekst)
    polowa = dlugosc // 2
    for i in range(polowa):
        if tekst[i] != tekst[dlugosc -i -1]:
            return False
    else:
        return True


# przykład wywołania

print(palindrom_petla("kobyłamamałybok"))
True
print(palindrom_petla("możejutrotadamasamadatortujeżom"))
True
print(palindrom_petla("Kobyła ma mały bok"))
False
print(palindrom_petla("test"))
False
```

Zapiszemy funkcję realizującą to samo zadanie z użyciem wyrażenia indeksującego. Zwróćmy uwagę, o ile krótsza jest teraz lista instrukcji:

```
def palindrom_indeks(tekst):
    return tekst == tekst[::-1]

```

```
# przykład wywołania
print(palindrom_indeks("kobyłamamałybok"))
True
print(palindrom_indeks("możejutrotadamasamadatortujeżom"))
True
print(palindrom_indeks("Kobyła ma mały bok"))
False
print(palindrom_indeks("test"))
False
```

**Ważne!**

Pamiętajmy, że wielkość liter i znaki przestankowe mają znaczenie podczas sprawdzania, czy wyrażenie jest palindromem. Działanie pierwszej z przedstawionych funkcji polega na porównywaniu odpowiadających sobie elementów po obu stronach ciągu. Ludzki mózg potrafi zignorować znaki, które uznaje za zbędne i brać pod uwagę wyłącznie litery (a także nie uwzględniać ich rozmiaru). Komputer nie jest w stanie tego zrobić.

Druga funkcja porównuje dwa ciągi od początku.

Jak zatem należy zdefiniować funkcję, aby móc sprawdzać bardziej skomplikowane łańcuchy znaków? Musimy najpierw pozbyć się takich elementów, jak spacja, średnik, przecinek, dwukropek, kropka, wykrzyknik lub znak zapytania, a następnie zamienić wielkie litery na małe albo małe na wielkie (czyli sprawić, że wszystkie znaki będą jednakowej wielkości). Dopiero wówczas możemy zająć się badaniem ciągu.

**Polecenie 1**

Zdefiniuj funkcję `sprawdz_palindrom(tekst)`, która będzie działać zgodnie z przedstawionymi założeniami i wykorzysta wyrażenia indeksujące.

**Dla zainteresowanych**

Aby uprościć zapis naszej funkcji, możemy wykorzystać wyrażenia regularne. Ze stosowanym w tym celu parametrem `r"[!?, ;:.]"` zapoznamy się dokładnie w oficjalnej dokumentacji języka Python i modułu `re`.

```
def palindrom_indeks_re(tekst):
    import re
    tekst = re.subn(r"[ ;,:.!?]","",tekst)[0]
    tekst = tekst.lower()

```

```
    return tekst == tekst[::-1]

# przykład testowania
print(palindrom_indeks_re("Kobyła; ma. mały bok!"))
True
```

## Tworzenie łańcuchów znaków `f-string`

W poprzednich e-materiałach dowiedzieliśmy się, w jaki sposób wyświetlać napisy lub inne obiekty na ekranie. Korzystaliśmy przy tym z funkcji `print()` oraz z metody `format()`.

Język Python, począwszy od wersji 3.5, oferuje dodatkowy mechanizm formatowania o nazwie **f-string**. Jest to ułatwiająca tworzenie kodu modyfikacja sposobu wywołania funkcji `print()`. Oto przykład jej zastosowania:

```
zmienna = 123
print(f'Napis na ekran z {zmienna} do podstawienia.')

Napis na ekran z 123 do podstawienia.
```

Jak widzimy, przed apostrofem lub cudzysłowem otwierającym ciąg znaków do wyświetlenia należy wpisać literę `f`. W nawiasach klamrowych wewnątrz ciągu trzeba natomiast podać nazwę obiektu, który zostanie wstawiony w to miejsce.

**Przykład 2**

Python dba o konwersję typu obiektów w ramach `f-string` (nie musimy zamieniać typów `int` czy `float` na `str`). Sprawdźmy nowy sposób wywoływania funkcji `print()`:

```
from math import sqrt
rok = 2019
urodzony = 1974
lat = rok - urodzony
pelnoletni = lat -18
pi = 3.141592
r = 19.53
obwod = 2 * pi * r
bok1 = 20
bok2 = 14.68
przekatna = sqrt(bok1**2 + bok2**2)
```

```
print("Oto nasze obliczenia:")
print(f"Mamy rok {rok}. Osoba urodzona w roku {urodzony} ma {la
print(f"Jest pełnoletnia od {pelnoletni} lat.")
print("-----------------------------------------")
print(f"Jest okrąg o promieniu {r} - jego obwód wynosi {obwod}.
print(f"Dany jest trójkąt prostokątny o bokach a={bok1} i b={bo
print(f"Długość boku c wg wzoru Pitagorasa wynosi {przekatna}."
```

```
Oto nasze obliczenia:
Mamy rok 2019. Osoba urodzona w roku 1974 ma 45 lat.
Jest pełnoletnia od 27 lat.
-----------------------------------------
Jest okrąg o promieniu 19.53 - jego obwód wynosi 122.7105835200
Dany jest trójkąt prostokątny o bokach a=20 i b=14.68.
Długość boku c wg wzoru Pitagorasa wynosi 24.80932082907551.
```

Użycie przedstawionego sposobu zapisu sprawia, że programiście łatwo jest przeanalizować kod programu.

**Polecenie 2**

Zdefiniuj funkcję `prostokat(bok_a, bok_b)`, która obliczy obwód, pole powierzchni oraz długość przekątnej prostokąta o podanych wymiarach, a następnie poda wszystkie te informacje, wykorzystując formatowanie `f-string`.

**Dla zainteresowanych**

Informacje na temat formatowania z zastosowaniem mechanizmu `f-string` znajdziemy w dokumencie [PEP-498].

**Problem 1**

Napisz program sprawdzający, czy dany napis jest palindromem.

**Specyfikacja:**

*Dane:*

- `zdanie` – łańcuch znaków

*Wynik:*

- Komunikat *Napis jest palindromem* lub *Napis nie jest palindromem*.

# Słownik

**wyrażenie indeksujące**

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

**wyrażenia regularne**

wzorzec pomocny podczas opisywania znaków w łańcuchu tekstowym; w języku Python dostępny w module standardowym `re`

# Animacja

**Polecenie 1**

Zapoznaj się z animacją prezentującą palindromy w genetyce.

Jeśli chcesz przypomnieć sobie najważniejsze informacje na temat budowy DNA, przejdź do e-materiału z biologii: DNA jako nośnik informacji genetycznej.

**Polecenie 2**

Poszukaj informacji na temat innych przykładów wykorzystania algorytmów wyszukiwania palindromów.


Film dostępny pod adresem /preview/resource/Rajb11V9P3m7S



Film nawiązujący do treści materiału: Palindromy w języku Python.

**Polecenie 3**

Zapisz notatkę, w której podsumujesz najważniejsze informacje zaprezentowane w animacji.

# Sprawdź się

Pokaż ćwiczenia: 

**Ćwiczenie 1**

Napisz funkcję `czy_palindrom`, która zwróci `True`, jeśli podany ciąg znaków jest palindromem, lub `False`, jesli nim nie jest. Nie używaj algorytmu opartego na odwracaniu napisu.

Twój program powinien ignorować wielkość liter. Sprawdzany tekst stanowi jeden wyraz,

**Specyfikacja:**

*Dane:*

- *tekst* – zmienna typu `string`

*Wynik:*

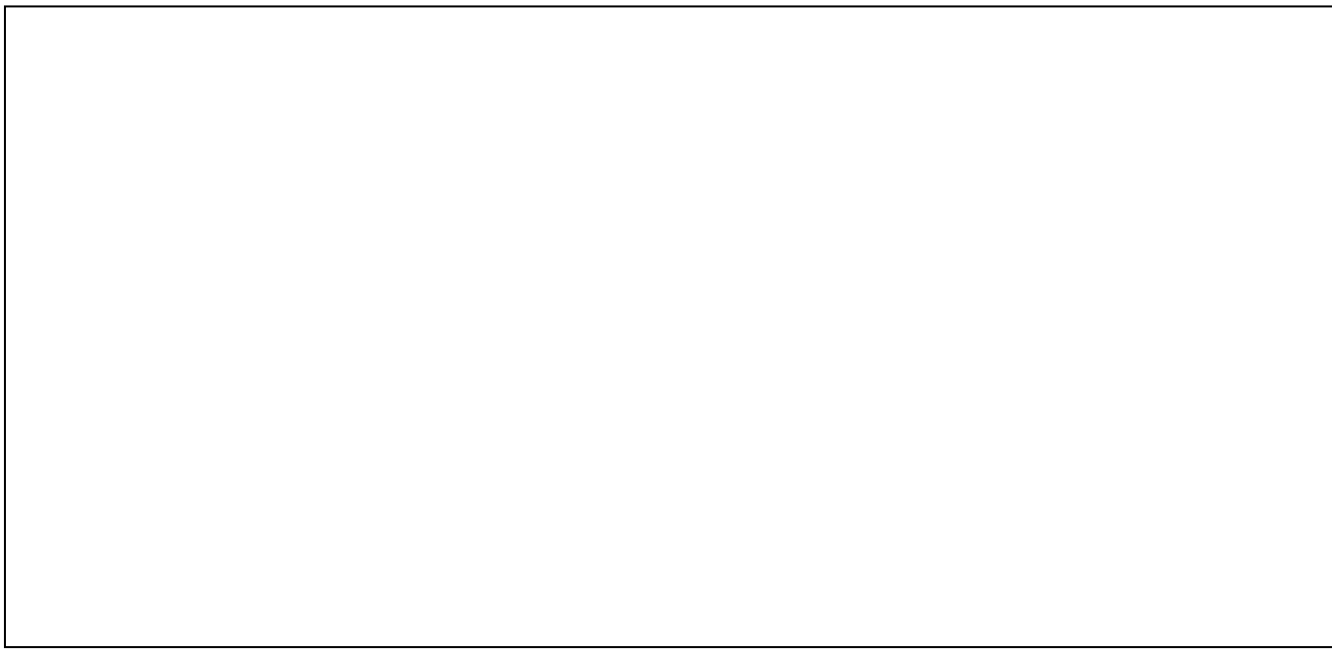
Wartość logiczna `True` lub `False`.

### Twoje zadania

1. Zdefiniowanie funkcji `czy_palindrom`.
2. Funkcja zwraca `True` dla ciągu `kajak`.

```
def czy_palindrom(tekst):
    # Tu uzupełnij kod
    pass
```

```
1
```



**Ćwiczenie 2**

Napisz funkcję `czy_palindrom`, która zwróci `True`, jeśli podany ciąg znaków jest palindromem, lub `False`, jeśli nim nie jest. Funkcja powinna ignorować przecinki i znaki spacji występujące w danych wejściowych. Powinna również, przed przystąpieniem do sprawdzenia, czy ciąg jest palindromem, zmiennić wszystkie wielkie litery na małe. Sprawdź swój program dla łańcucha znaków *a wart wór kota, to krów trawa.*

**Specyfikacja:**

*Dane:*

- *wyrazenie* – zmienna typu `string`

*Wynik:*

Wartość logiczna `True` lub `False`.

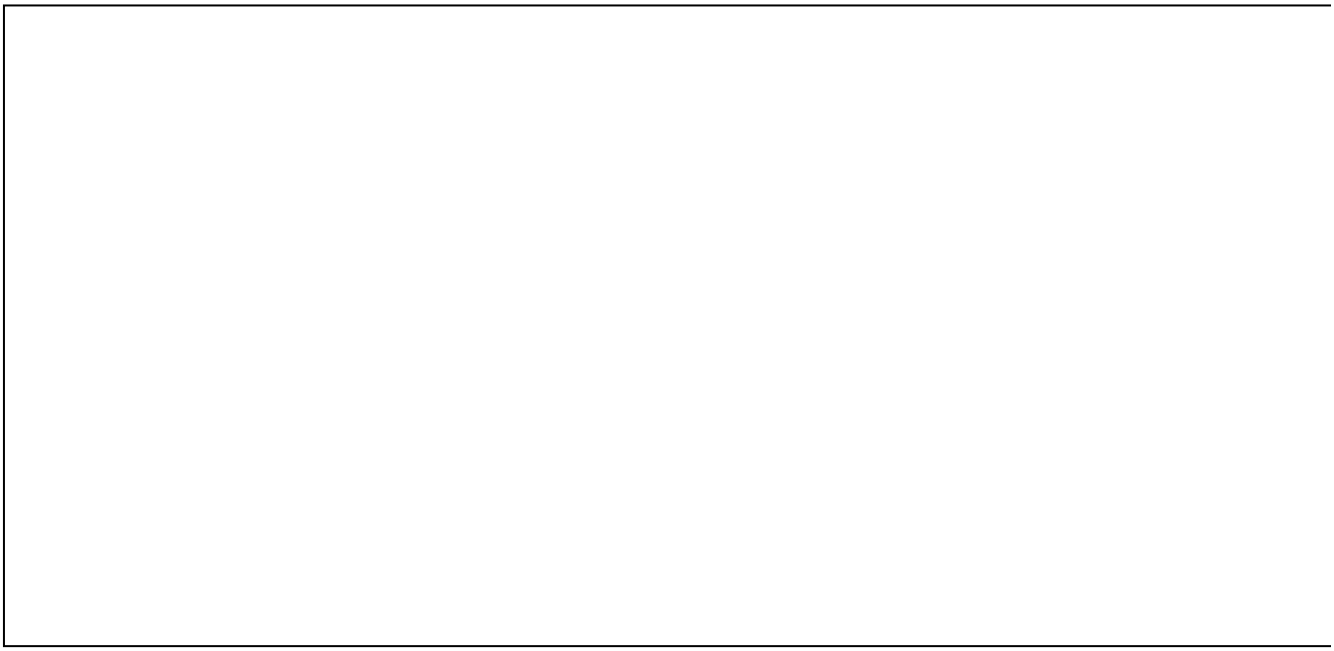
## Twoje zadania

1. Zdefiniowanie funkcji `czy_palindrom`
2. Funkcja zwraca `True` dla ciągu `A wart wór kota, to krów trawa.`

```
def czy_palindrom(wyrazenie):
    # Tu uzupełnij kod
    pass

wyrazenie = "A wart wór kota, to krów trawa"
```

```
1
```



**Ćwiczenie 3**

Pewna firma z branży lotniczej miała problem z przekłamaniami transmisji danych – zdarzało się, że urządzenie nadawcze wysyłało bit 1, który jednak był interpretowany przez odbiornik jako 0. Uznano, że rozwiązaniem tego problemu będzie zastosowanie następującego kodu: po każdych ośmiu bitach nadawane są te same bity, ale w odwrotnej kolejności. Dzięki takiemu rozwiązaniu można określić, czy otrzymane dane są poprawne. Napisz program, który określi, czy podany ciąg zer i jedynek jest poprawnym kodem (ma poprawną strukturę oraz poprawną długość). Dla prawidłowych kodów powinien drukować wiadomość *Poprawny kod*, a dla nieprawidłowych: *Niepoprawny kod*. Przetestuj jego działanie dla ciągu bitów *1001110110111001*.

**Specyfikacja:**

*Dane:*

- *dane* – łańcuch znaków

*Wynik:*

Program wyświetla komunikat *Poprawny kod* lub *Niepoprawny kod*.

### Twoje zadania

1. Zdefiniowanie funkcji `czy_poprawny_kod`.
2. Funkcja zwraca komunikat `Poprawny kod` dla kodu `1001110110111001`.

```
def czy_poprawny_kod(dane):
    pass

dane = "1001110110111001"
```

```
1
```

# Dla nauczyciela

**Autor:** Zespół autorski Contentplus.pl sp. z o.o.
**Przedmiot:** Informatyka

**Temat: Palindromy 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.

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,

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 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):**

- Zastosujesz algorytm sprawdzający, czy dany ciąg znaków jest palindromem, wykorzystując pętlę `for`.
- Użyjesz wyrażeń indeksujących do sprawdzenia, czy ciąg znaków jest palindromem.
- Przeanalizujesz nowy sposób formatowania napisów – `f-string`.

**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: „Palindromy w języku Python”. Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj”.

**Faza wstępna:**

1. Nauczyciel wyświetla i odczytuje temat lekcji oraz cele zajęć. Prosi uczniów o sformułowanie kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

**Faza realizacyjna:**

1. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Animacja". Na forum klasy uczniowie analizują Polecenia 1 i 2.
2. Nauczyciel dzieli klasę na dwie grupy. Pierwsza z nich analizuje Przykład 1 i wykonuje Polecenie 1 z sekcji „Przeczytaj", a druga analizuje Przykład 2 i wykonuje Polecenie 2. Przedstawiciele grup przedstawiają rozwiązania na forum klasy.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1-2 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

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

**Praca domowa:**

1. Uczniowie wykonują ćwiczenie 3 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:**

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj", „Animacja", „Sprawdź się" jako materiał do lekcji powtórkowej.