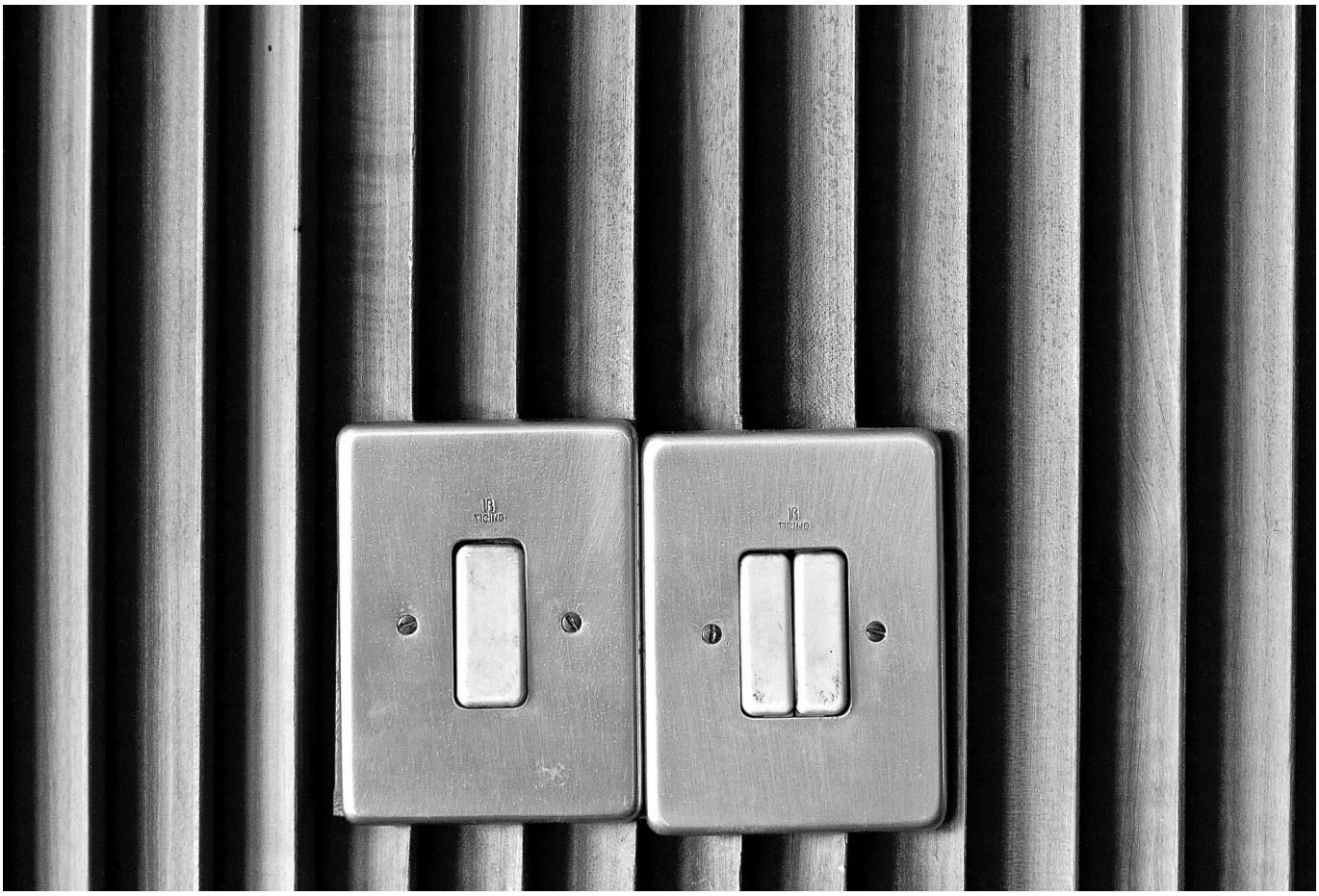


Wybór wielokrotny w języku Python

Źródło: Karim Manjra, domena publiczna.

Instrukcja warunkowa sprawdza się bardzo dobrze, gdy trzeba wybrać do zrealizowania jeden z dwóch scenariuszy opisanych w kodzie programu. Sprawy komplikują się wraz ze zwiększaniem liczby scenariuszy. Lepszym rozwiązaniem okazuje się wówczas instrukcja wielokrotnego wyboru.

W tym e-materiale omówimy także słowniki, dzięki którym możemy w języku Python wykonać konstrukcję wielokrotnego wyboru.

Podstawowe informacje na temat instrukcji wielokrotnego wyboru zostały omówione w e-materiale Wybór wielokrotny.

Ciekawi cię, jak wygląda implementacja tego algorytmu w innych językach programowania? Możesz się z nimi zapoznać w pozostałych e-materiałach z tej serii:

- Wybór wielokrotny w języku C++,

- Wybór wielokrotny w języku Java.

> **Twoje cele**
>
> - Użyjesz wyrażenia trójargumentowego (`a if b else c`).
> - Przeanalizujesz słownikowy typ danych (`dict`).
> - Scharakteryzujesz różnice między słownikiem a listą.
> - Użyjesz słownika jako alternatywy dla konstrukcji wielokrotnego wyboru.
> - Dokonasz analizy instrukcji `match/case`.

Przeczytaj

Wielokrotny wybór w języku Python

Podczas tworzenia programu zdarzają się sytuacje, w których potrzebujemy zrealizować jeden z wielu scenariuszy działań.

Przeanalizujmy przykład, gdzie mamy wypisać informację, w którym z następujących przedziałów: $(10,\ 15]$, $(15,\ 21]$, $(21,\ 50]$ zawiera się dana liczba.

Do zrealizowania takiego scenariusza możemy zastosować konstrukcję `if/elif/else`, ponieważ wybieramy jeden z wielu przedziałów.

Ogólną postać instrukcji znajdziesz w e-materiale Instrukcja warunkowa w języku Python.

> **Przykład 1**
>
> Napiszmy program, który zrealizuje taki scenariusz.

```
wartosc = 23

if 10 < wartosc and wartosc <= 15:
    print("Przedział 1")
elif wartosc > 15 and wartosc <= 21:
    print("Przedział 2")
elif wartosc > 21 and wartosc <= 50:
    print("Przedział 3")
else:
    print("Liczba nie zawiera się w żadnym z przedziałów")
```

> **Ważne!**
>
> Powyższy program możemy zapisać, korzystając z kilku instrukcji warunkowych.

```
wartosc = 23

if wartosc > 10 and wartosc <= 15:
    print("Przedział 1")
if wartosc > 15 and wartosc <= 21:
    print("Przedział 2")
if wartosc > 21 and wartosc <= 50:
    print("Przedział 3")
if wartosc <= 10 or wartosc > 50:
    print("Liczba nie zawiera się w żadnym z przedziałów")
```

Przy kilku instrukcjach `if` warunek każdej z nich sprawdzany jest niezależnie od innych, natomiast przy konstrukcji `if/elif/else`, jeśli jakiś warunek zostanie spełniony, to dalsze warunki nie są sprawdzane.

Przykład 2

Spróbujmy teraz przeanalizować inną sytuację. Chcemy, w zależności od podanej przez użytkownika liczby (wybranej opcji), wykonać pewną akcję. Zdefiniujemy cztery funkcje, a do wyboru odpowiedniej opcji wykorzystamy instrukcję `if/elif/else`:

```
def funkcja_1():
    print("Funkcja 1.")

def funkcja_2():
    print("Funkcja 2.")

def funkcja_3():
    print("Funkcja 3.")

def funkcja_x():
    print("Funkcja x.")


```

```
wybor = input("Podaj numer: ")
wynik = int(wybor)

if wynik == 1:
    funkcja_1()
elif wynik == 2:
    funkcja_2()
elif wynik == 3:
    funkcja_3()
else:
    funkcja_x()

# Efekt wykonania:

# Podaj numer: 2
# Funkcja 2.
```

Użycie słownika jako alternatywy dla konstrukcji wielokrotnego wyboru `if/elif/else`

Możemy skorzystać z pewnej właściwości języka Python. Zgodnie z nią każda funkcja może być potraktowana jako obiekt i przypisana do dowolnego elementu innego obiektu. Zatem poprawny będzie następujący przykład.

Zdefiniujmy cztery funkcje i wykonajmy je w zależności od wyboru użytkownika.

```
def funkcja_1():
    print("Funkcja 1.")

def funkcja_2():
    print("Funkcja 2.")

def funkcja_3():
    print("Funkcja 3.")

```

```
def funkcja_x():
    print("Funkcja x.")


wybor = input("Podaj numer: ")
wynik = int(wybor)

# tworzymy słownik zawierający jako wartości nazwy funkcji
slownik = {1: funkcja_1, 2: funkcja_2, 3: funkcja_3}

# jeśli wybór jest kluczem słownika, wykonujemy funkcję
if wynik in slownik:
    slownik[wynik]()
# w innym przypadku wykonujemy konkretną funkcję
else:
    funkcja_x()
```

Trójargumentowe wyrażenie wyboru

Używane jest w sytuacjach, gdy wystarczy zwrócić prostą informację zależną od jednego warunku. W takim przypadku rozbudowywanie kodu jest niecelowe.

Oto wyrażenie trójargumentowe zapisane w języku Python:

```
wartość_gdy_prawda if warunek_logiczny else wartość_gdy_fałsz
```

Instrukcja taka odpowiada poleceniom:

```
if warunek_logiczny:
    wartość_gdy_prawda
else:
    wartość_gdy_fałsz
```

Porównajmy kody dwóch funkcji. Mają one sprawdzić, czy tekst przekazany im jako argument jest palindromem. Sprawdzenie, czy słowo jest palindromem, wykonane jest przy użyciu wyrażenia indeksującego.

Pierwsza funkcja wykorzystuje zwykłą konstrukcję `if/else`:

```
def czy_palindrom(tekst):
    if tekst == tekst[::-1]:
        return True
    else:
        return False
```

Druga funkcja wykorzystuje wyrażenie trójargumentowe:

```
def czy_palindrom(tekst):
    return True if tekst == tekst[::-1] else False
```

Obie powyższe funkcje mogą być zapisane również następująco:

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

Innym przykładem zastosowania wyrażenia trójargumentowego jest podanie jednego z dwóch komunikatów, zależnych od spełnienia pewnego warunku. Posłużymy się funkcją `print()`:

```
wartosc = 15
print('Dużo' if wartosc > 20 else 'Mało', 'pieniędzy.')
# Mało pieniędzy.

wartosc = 23
print('Dużo' if wartosc > 20 else 'Mało', 'pieniędzy.')
# Dużo pieniędzy.
```

Instrukcja `match/case`

W wersji 3.10 języka Python została wprowadzona instrukcja `match/case`. Instrukcja ta jako argument przyjmuje wyrażenie i porównuje jego wartość ze zdefiniowanymi schematami. Jeśli wyrażenie jest zgodne ze zdefiniowanym schematem, wykonywane są związane z nim instrukcje.

Instrukcja `match/case` zapisana za pomocą języka Python wygląda następująco:

```
match numer_ucznia:
    case 1:
        # wypisz informacje o uczniu numer 1.
    case 2:
        # wypisz informacje o uczniu numer 2.
    case 3:
        # wypisz informacje o uczniu numer 3.
    case 4:
        # wypisz informacje o uczniu numer 4.
```

```
numerUucznia = 3

match numer_ucznia:
    case 1:
        # wypisz informacje o uczniu numer 1.
    case 2:
        # wypisz informacje o uczniu numer 2.
    case 3:
        # wypisz informacje o uczniu numer 3.
    case 4:
        # wypisz informacje o uczniu numer 4.
```

Instrukcja `match/case` pobiera obiekt (`numer_ucznia`), testuje go względem jednego lub więcej wzorców dopasowania (`case 1`, `case 2` etc.) i wykonuje akcję, jeśli znajdzie dopasowanie.

Po każdym słowie kluczowym `case` następuje wzorzec dopasowania.

Python wykonuje dopasowania, przechodząc przez listę przypadków od góry do dołu. Przy pierwszym dopasowaniu Python wykonuje instrukcje w odpowiadającym im bloku przypadków, a następnie przeskakuje na koniec bloku dopasowania i kontynuuje dalszą część programu. Oznacza to, że po znalezieniu pierwszego pasującego przypadku, kończy się działanie instrukcji `match/case`.

Załóżmy, że chcemy napisać program, który wyświetli komunikat `Wakacje!`, jeśli użytkownik poda numer wakacyjnego miesiąca (zakładamy, że wakacyjne miesiące to lipiec i sierpień) lub `To nie wakacje :-(`, jeśli poda numer jakiegokolwiek innego miesiąca.

```
miesiac = 2

match miesiac:
    case 1:
        print("To nie wakacje :-(")
    case 2:
        print("To nie wakacje :-(")
    case 3:
        print("To nie wakacje :-(")
    case 4:
        print("To nie wakacje :-(")
    case 5:
        print("To nie wakacje :-(")
    case 6:
        print("To nie wakacje :-(")
    case 7:
        print("Wakacje!")
    case 8:
        print("Wakacje!")
    case 9:
        print("To nie wakacje :-(")
    case 10:
```

```
        print("To nie wakacje :-(")
    case 11:
        print("To nie wakacje :-(")
    case 12:
        print("To nie wakacje :-(")
```

Łączenie warunków

Używając operatora |, oznaczającego alternatywę, możemy łączyć warunki.

```
miesiac = 2

match miesiac:
    case 1 | 2 | 3 | 4 | 5 | 6 | 9 | 10 | 11 | 12:
        print("To nie wakacje :-(")
    case 7 | 8:
        print("Wakacje!")
```

Domyślny przypadek

W instrukcji `match/case` możemy podać przypadek _, który oznacza „w każdym innym przypadku", czyli jeżeli wartość zmiennej podanej do sprawdzenia nie będzie równa żadnemu z ustawionych przypadków, program wykona fragment kodu zawarty w poleceniu.

Przeanalizuj przykład:

```
miesiac = 27

match miesiac:
    case 1 | 2 | 3 | 4 | 5 | 6 | 9 | 10 | 11 | 12:
        print("To nie wakacje :-(")
    case 7 | 8:
        print("Wakacje!")
    case _:

```

```
        print("Wybrano liczbę spoza przedziału [1, 12]")
```

Zaawansowane wykorzystanie

Instrukcja `match/case` pozwala na zdefiniowanie zmiennych lokalnych, które będą dopasowywane do argumentu. Do tych zmiennych lokalnych możemy się odwołać w zasięgu pojedynczej klauzuli `case`. Co więcej, możemy doprecyzować, kiedy ma się do tych zmiennych wpasować, stosując polecenie `if`.

Przeanalizujmy następujący program:

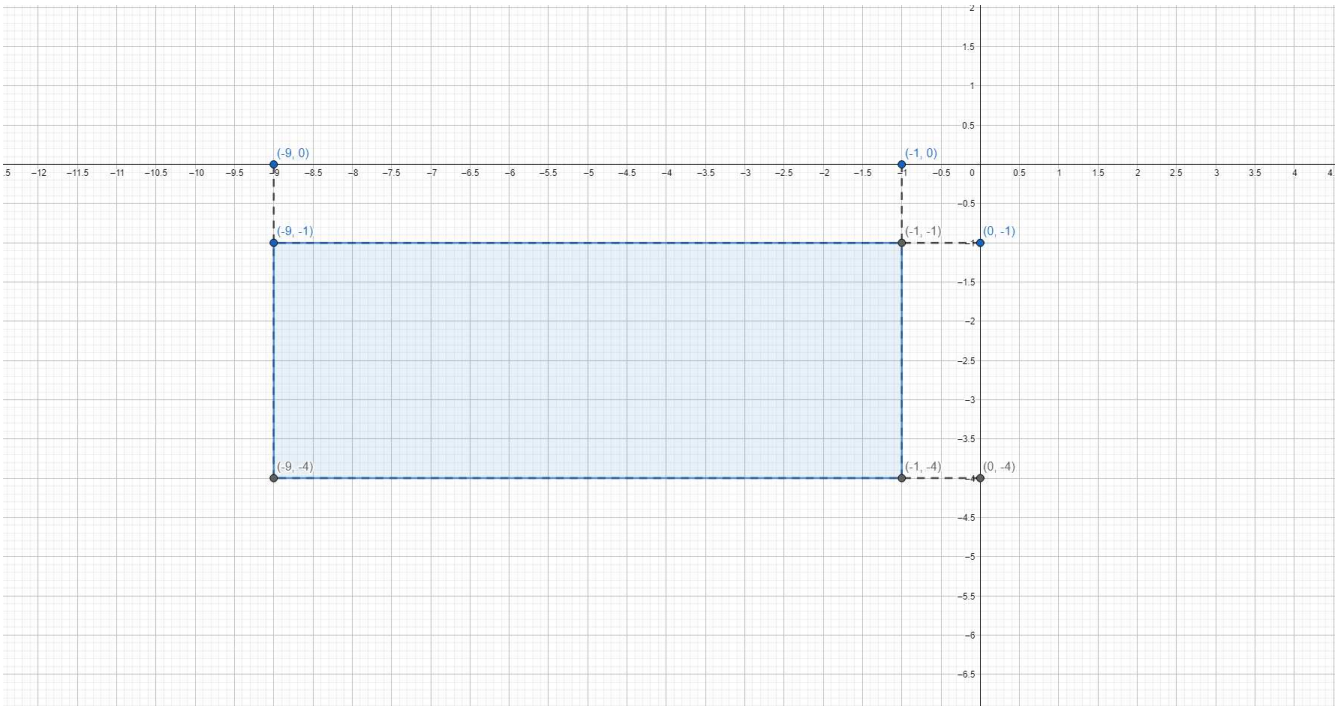
```
def wypisz_informacje(punkt, dziedzina_x=(-9, -1), dziedzina_y=(-
    x1, x2 = dziedzina_x
    y1, y2 = dziedzina_y

    match punkt:
        case (x, y) if x1 < x < x2 and y1 < y < y2:
            print("Punkt: ", punkt, " należy do rozważanej dziedz
        case (0, 0):
            print("Punkt: ", punkt, " nie należy do dziedziny, al
        case (x, 0):
            print("Punkt: ", punkt, " nie należy do dziedziny, al
        case (0, y):
            print("Punkt: ", punkt, " nie należy do dziedziny, al
        case (_, _):
            print("Punkt: ", punkt, "nie należy do dziedziny")
```

Funkcja `wypisz_informacje` przyjmuje trzy parametry: `punkt`, `dziedzina_x`, `dziedzina_y`. Dwa ostatnie mają podane wartości domyślne. Jeśli przy wywoływaniu funkcji do argumentów nie zostaną przekazane nowe wartości, funkcja zostanie wywołana z wartościami domyślnymi.

Widać to przy wywołaniach funkcji – funkcja `wypisz_informacje` wywoływana jest z jednym argumentem (dwa pozostałe mają przypisane wartości domyślne).

Wartości przekazane do funkcji są krotkami.

W programie definiujemy pewną dziedzinę, która ograniczona jest podanymi wartościami. Zwizualizujmy ją.



Moglibyśmy zdefiniować dwa główne przypadki:

- Punkt należy do rozważanej dziedziny.
- Punkt nie należy do rozważanej dziedziny.

Jeśli punkt miałby należeć do dziedziny, wartość współrzędnej x musiałaby mieścić się w następującym przedziale $[-9, \ -1]$, a wartość współrzędnej y musiałaby zawierać się w następującym przedziale $[-4, \ -1]$:

```
case (x, y) if x1 < x < x2 and y1 < y < y2:
```

Gdyby punkt nie znajdował się w tym przedziale, nie należałby do dziedziny:

```
case (x, y) if not (x1 < x < x2 and y1 < y < y2):
```

Jednak to niejedyny przypadek, jaki możemy zdefiniować. Pozostałe:

- Punkt leży na osi rzędnych.
- Punkt leży na osi odciętych.
- Punkt leży w środku układu współrzędnych.

Jeśli punkt miałby leżeć na osi rzędnych, jego pierwsza współrzędna musiałaby wynosić 0.

```
case (0, y):
```

Jeśli punkt miałby leżeć na osi odciętych, jego druga współrzędna musiałaby wynosić 0.

```
case (x, 0):
```

Jeśli punkt miałby leżeć w środku układu współrzędnych, obie jego współrzędne musiałyby wynosić 0.

```
case (0, 0):
```

Wiemy, że w przypadku instrukcji `match/case` program kończy działanie instrukcji, kiedy zostanie spełniony pierwszy warunek. Musimy zatem odpowiednio zaplanować ułożenie warunków.

Zacznijmy od sprawdzenia, czy punkt należy do dziedziny. Zatem to ten warunek zostanie zapisany jako pierwszy.

Następnie sprawdźmy, czy punkt leży w środku układu współrzędnych, na rzędnej lub odciętej. Zapiszmy kolejne warunki.

Warto pamiętać, że trzy wspomniane wcześniej przypadki odnoszą się do sytuacji, w której punkt nie należy do dziedziny, ale za to leży na osi współrzędnych. To

rozróżnienie również powinno znaleźć się w komunikatach wyświetlanych przez program.

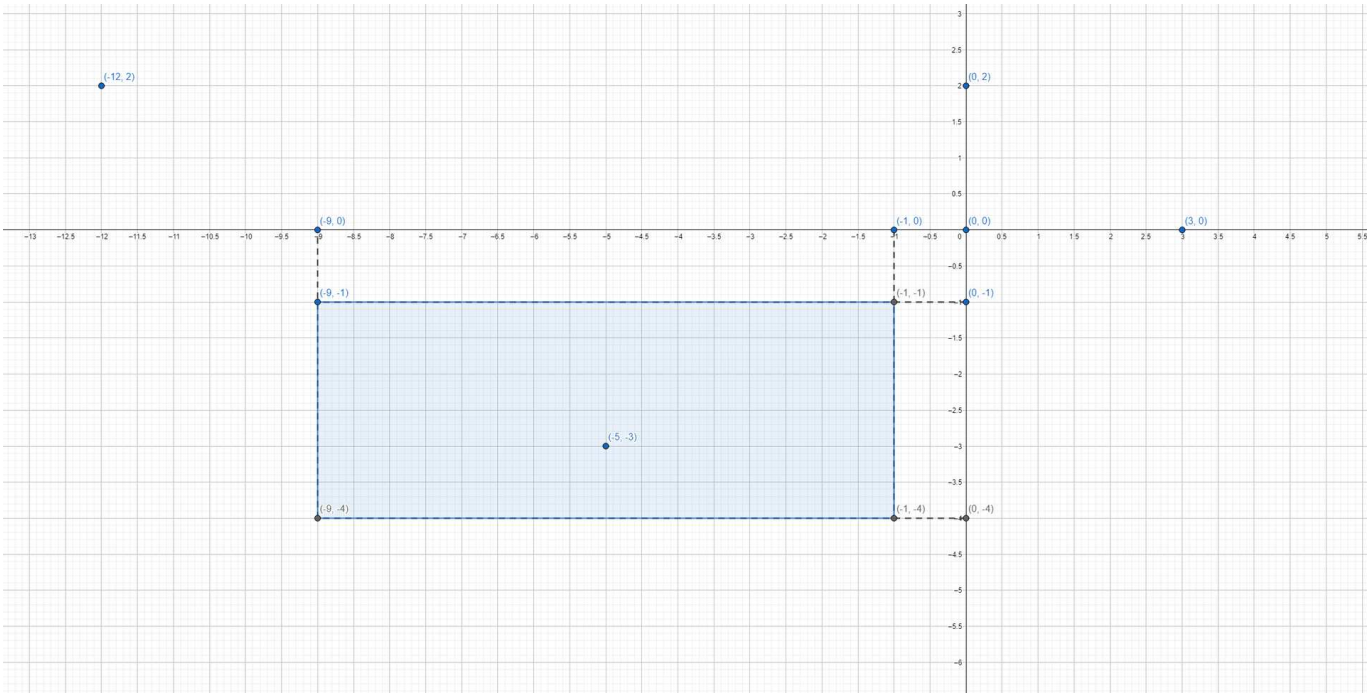
Sprawdziliśmy kilka konkretnych przypadków. Weźmy jednak pod uwagę również taką sytuację, w której punkt nie należy do dziedziny, nie leży w środku układu współrzędnych ani na żadnej osi.

```
def wypisz_informacje(punkt, dziedzina_x=(-9, -1), dziedzina_y=(-
    x1, x2 = dziedzina_x
    y1, y2 = dziedzina_y

    match punkt:
        case (x, y) if x1 < x < x2 and y1 < y < y2:
            print("Punkt: ", punkt, " należy do rozważanej dziedz
        case (0, 0):
            print("Punkt: ", punkt, " nie należy do dziedziny, al
        case (x, 0):
            print("Punkt: ", punkt, " nie należy do dziedziny, al
        case (0, y):
            print("Punkt: ", punkt, " nie należy do dziedziny, al
        case (_, _):
            print("Punkt: ", punkt, "nie należy do dziedziny")
```

Przetestujmy działanie programu dla następujących punktów:

```
-5, -3
-12, 2
0, 0
3, 0
0, 2
```

Nanieśmy je na wizualizację:



Wywołajmy funkcję z nimi jako argumentami:

```
wypisz_informacje((-5, -3))
wypisz_informacje((-12, 2))
wypisz_informacje((0, 0))
wypisz_informacje((3, 0))
wypisz_informacje((0, 2))
```

Wynik programu:

```
Punkt:  (-5, -3)  należy do rozważanej dziedziny
Punkt:  (-12, 2) nie należy do dziedziny
Punkt:  (0, 0)  nie należy do dziedziny, ale jest położony w środ
Punkt:  (3, 0)  nie należy do dziedziny, ale jest położony na osi
Punkt:  (0, 2)  nie należy do dziedziny, ale jest położony na osi
```

> **Polecenie 1**
> Porównaj wynik programu z wizualizacją.

Słownik

obiekt niezmienny

(ang. *immutable*) obiekt, który po stworzeniu nie może być modyfikowany.

palindrom

ciąg znaków, który brzmi tak samo czytany w obu kierunkach (od strony lewej do prawej i wspak)

wyrażenie indeksujące

(ang. *slice*) zawarte w nawiasach kwadratowych wyrażenie, które pozwala określić wybraną część sekwencji; może ono mieć postać `[start:koniec:krok]`; wyrażenie indeksujące zwraca wycinek, zaczynający się od argumentu `start`, następnie zwiększając indeks o `krok`, aż do przekroczenia lub wyrównania wartości `koniec`; nieprzekazanie argumentu `start` skutkuje tym, że zaczynamy wycinek od początku sekwencji; podobnie, gdy nie przekażemy argumentu `koniec`, to górnym ograniczeniem wycinka będzie ostatnia wartość; wartość domyślna dla parametru `krok` to jeden; lista w języku *Python* może być indeksowana od tyłu; ostatnia wartość listy jest dostępna pod indeksem -1, przedostatnia pod -2 itd.; argument `krok` również może przyjąć wartość ujemną, wtedy wartość domyślna parametru `start` jest równa -1, np. dla `[::-1]` otrzymamy listę w odwrotnej kolejności

Schemat interaktywny

Polecenie 1
Napisz program, który wyświetla informację zwrotną w zależności od oceny otrzymanej ze sprawdzianu.

Specyfikacja problemu:

Dane:

- ocena – liczba naturalna z przedziału: $[1,\ 6]$

Wynik:

Komunikat dopasowany do danej oceny:

- Ocena 6: `"Wspaniale!"`
- Oceny 5 i 4: `"Bardzo dobrze"`
- Ocena 3: `"Spróbuj powtórzyć najważniejsze informacje"`
- Ocena 2: `"Podejdź do sprawdzianu jeszcze raz"`
- Ocena 1:
  `"Ten materiał sprawił ci trudność, może porozmawiaj na ten te`

```
ocena = 5    # dana ocena - dla testów zmień jej wartość

### tutaj dopisz kod
```

```
1
```

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/R1Js8JvufqhCB

Film opowiadający o wyborze wielokrotnym w języku Python.

Plik o rozmiarze 317.00 B w języku polskim

Plik o rozmiarze 325.00 B w języku polskim

Plik o rozmiarze 503.00 B w języku polskim

Plik o rozmiarze 584.00 B w języku polskim

Polecenie 3

Gen niebieskich oczu to gen recesywny, co oznacza, że jego działanie ujawnia się tylko w sytuacji, kiedy wystąpią jego dwie kopie. Uczniowie napisali program, który na podstawie koloru oczu rodziców ustala, jakie jest prawdopodobieństwo tego, że dziecko również będzie miało niebieskie oczy.

Jeśli dwójka rodziców ma niebieskie oczy, szansa na to, że dziecko będzie miało niebieskie oczy, są **bardzo wysokie**.

W sytuacji, kiedy tylko jeden z rodziców ma niebieskie oczy, te szanse są już **średnie**.

Kiedy żadne z rodziców nie ma niebieskich oczu, szanse na to, że dziecko będzie miało niebieskie oczy, są **niewielkie**.

Więcej na ten temat dowiesz się w e-materiale Rachunek prawdopodobieństwa w genetyce klasycznej.

Specyfikacja problemu:

Dane:

- `kolor1` – wartość logiczna; wartość `prawda` oznacza niebieski kolor oczu pierwszego rodzica
- `kolor2` – wartość logiczna; wartość `prawda` oznacza niebieski kolor oczu pierwszego rodzica

Wynik:

Program wypisuje szanse dziecka na odziedziczenie niebieskiego koloru oczu.

Zapoznaj się ze schematem i napisz ten program, używając języka Python. Przetestuj jego działanie dla rodziców, którzy nie mają niebieskich oczu.

```
# Tutaj dodaj własny kod. Użyj funkcji
# print()
```

```
```

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Utwórz prawdziwe zdania na temat słowników w języku Python, łącząc w pary właściwe wyrażenia.

Przechowują elementy	dwóch takich samych kluczy.
Umożliwiają przechowywanie	parami klucz-wartość.
Uniemożliwiają posiadanie	dwóch takich samych wartości.

Ćwiczenie 2

Napisz program, w którym zdefiniujesz funkcję `konwersja_na_rzymska(liczby)`, która dla liczb całkowitych z przedziału $[1,\ 20]$ zwróci liczbę zapisaną w systemie rzymskim. Dla liczb spoza tego przedziału funkcja powinna zwrócić ciąg znaków `***`. Przetestuj działanie programu, wywołując funkcję dla argumentów 12 oraz 22.

Uwaga! W module testowym nie zadziała instrukcja `match/case`, ponieważ nie jest w nim obsługiwany język Python w wersji 3.10.

Specyfikacja problemu:

Dane:

- `lista_argumentow` – lista liczb całkowitych

Wynik:

Program na standardowe wyjście wypisuje wynik funkcji `konwersja_na_rzymska(liczby)` dla każdego argumentu z listy `lista_argumentow` w osobnej linii.

Twoje zadania

1. Program wypisuje na standardowe wyjście liczbę 12 i 22 w systemie rzymskim (w osobnej linii).

```
def konwersja_na_rzymska(liczba):
    # Tutaj wpisz własny kod
    return None

lista_argumentow = [12, 22]
for argument in lista_argumentow:
    print(konwersja_na_rzymska(argument))
```

```
1
```

Ćwiczenie 3

Wykorzystując poznane informacje, zdefiniuj funkcję `pierwszy_znak_rzymskiej(liczba)`, która dla liczb całkowitych z przedziału $[1,\ 3999]$ zwróci pierwszy znak tej liczby w systemie rzymskim, zaś dla liczb spoza tego zakresu zwróci wartość `False`. Przetestuj kod dla argumentów znajdujących się w tablicy `lista_argumentow`.

Uwaga! W module testowym nie zadziała instrukcja `match/case`, ponieważ nie jest obsługiwany język Python w wersji 3.10.

Specyfikacja problemu:

Dane:

- `lista_argumentow` – lista liczb całkowitych

Wynik:

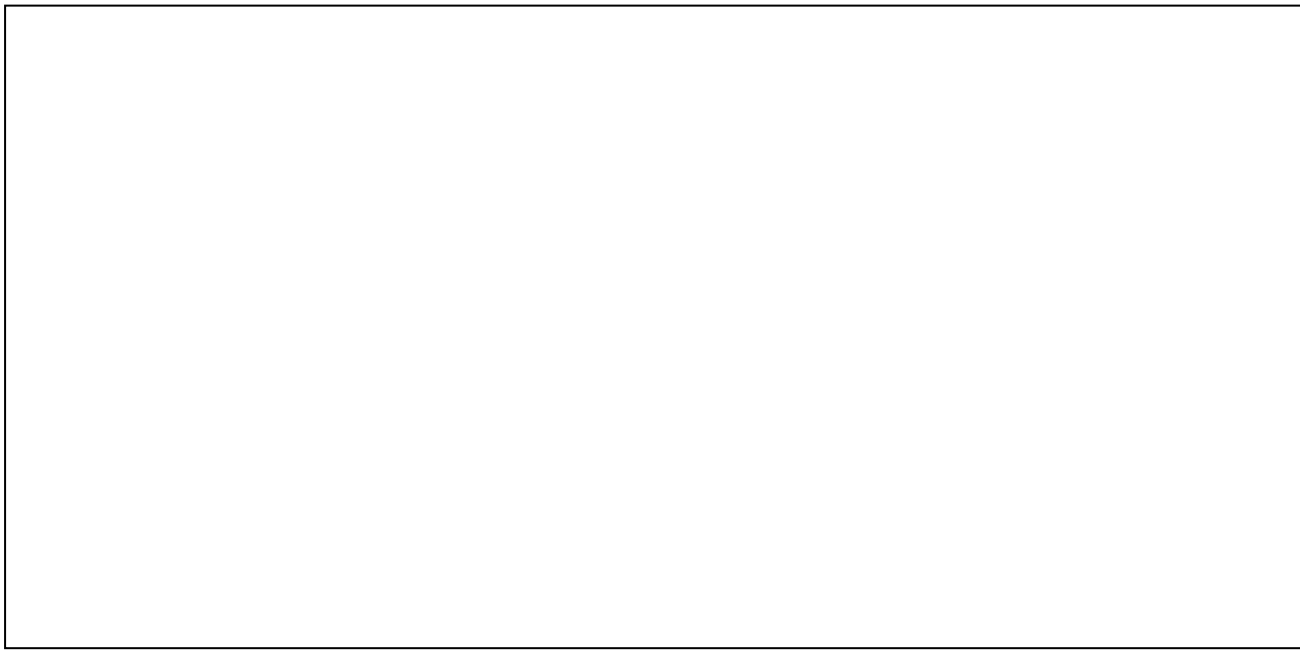
Program wypisuje na standardowe wyjście w kolejności występującej w liście `lista_argumentow` (w osobnej linii) pierwszy znak argumentu zamienionego na liczbę rzymską.

Twoje zadania

1. Program wypisuje na standardowe wyjście pierwszy znak każdej liczby znajdującej się w `lista_argumentow` po zamienieniu na rzymski system liczb. Wynik dla każdego argumentu znajduje się w osobnej linii.

```
def pierwszy_znak_rzymskiej(liczba):
    # tutaj Twój kod
    return None

lista_argumentow = [-5, 3, 10, 92, 389, 3998]
for argument in lista_argumentow:
    print(pierwszy_znak_rzymskiej(argument))
```

```
1
```



Ćwiczenie 4

Wskaż poprawną konstrukcję instrukcji `match/case`.

```
1
```

- [ ]
```
x = 2
match (x):
    case 2 || 3:
        print("Otrzymałem 2 lub 3.")
    case _:
        print("Otrzymałem coś innego niż 2 lub 3.")
```

- [ ]
```
x = 2
match (x):
    case 2 | 3:
        print("Otrzymałem 2 lub 3.")
    case else:
        print("Otrzymałem coś innego niż 2 lub 3.")
```

- [ ]
```
x = 2
match (x):
    case 2 or 3:
        print("Otrzymałem 2 lub 3.")
    case _:
        print("Otrzymałem coś innego niż 2 lub 3.")
```

- [ ]
```
x = 2
match (x):
    case 2 | 3:
        print("Otrzymałem 2 lub 3.")
    case _:
        print("Otrzymałem coś innego niż 2 lub 3.")
```

Ćwiczenie 5

Napisz program, który sprawdzi, czy pewna liczba należy do pięcioelementowego zbioru {5, 10, 15, 20, 25}. Jeżeli tak jest, należy wyświetlić komunikat `tak`. W przeciwnym wypadku powinien zostać wypisany komunikat `nie`.

Twoje zadania

1. Program sprawdza, czy liczba 20 należy do zbioru {5, 10, 15, 20, 25}.

```
x = 20

    // Tutaj dodaj własny kod.
    // Aby wypisać komunikat użyj polecenia print
```

```
```

Ćwiczenie 6

Napisz program, który sprawdzi, czy różnica między wartościami zmiennych `wiek_osoby_b` i `wiek_osoby_a` wynosi 1, 2 lub 3, czy też jest inna. W każdym z wymienionych przypadków należy wyświetlić odpowiedni komunikat.

Twoje zadania

1. Program sprawdza, ile wynosi różnica między wartościami zmiennych `wiek_osoby_b` i `wiek_osoby_a`, a następnie wyświetla stosowny komunikat.

```
wiek_osoby_a = 13
wiek_osoby_b = 15

    ## Tutaj dodaj własny kod.
    ## Wypisz "Osoby są w takim samym wieku", jeżeli różnica między wiek_osoby_b a wiek_osoby_a jest równa 0
    ## Wypisz "Różnica wynosi 1 rok", jeżeli różnica między wiek_osoby_b a wiek_osoby_a jest równa 1
    ## Wypisz "Różnica wynosi 2 lata", jeżeli różnica między wiek_osoby_b a wiek_osoby_a jest równa 2
    ## Wypisz "Różnica wynosi 3 lata", jeżeli różnica między wiek_osoby_b a wiek_osoby_a jest równa 3
    ## Wypisz "Różnica wieku wynosi więcej niż 3 lata", jeżeli różnica między wiek_osoby_b a wiek_osoby_a jest
```

```

```

Ćwiczenie 7

Małgosia otrzymuje wynagrodzenie za wykonywanie prac domowych. Każda czynność jest opłacana według określonej stawki. Napisz program, który na podstawie wysokości otrzymanej kwoty poinformuje, czym zajmowała się Małgosia.

Twoje zadania

1. Program ma za zadanie sprawdzić, jaką czynność wykonała Małgosia, jeżeli dostała 15 zł i wyświetlić odpowiedni komunikat.

```
wynagrodzenie = 15

    ## Tutaj dodaj własny kod.
    ## Wypisz "Małgosia podlała kwiatki", jeżeli wynagrodzenie jest równe 2
    ## Wypisz "Małgosia sprzątała kuchnię", jeżeli wynagrodzenie jest równe 5
    ## Wypisz "Małgosia umyła okna", jeżeli wynagrodzenie jest równe 7
    ## Wypisz "Małgosia posprzątała piwnicę", jeżeli wynagrodzenie jest równe 12
    ## Wypisz "Małgosia umyła naczynia", jeżeli wynagrodzenie jest równe 15
    ## Wypisz "Małgosia wyrwała chwasty w ogródku", jeżeli
```

```

```

Dla nauczyciela

Autor: Adam Jurkiewicz

Przedmiot: Informatyka

Temat: Wybór wielokrotny w języku Python

Grupa docelowa:

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

Podstawa programowa:

Cele kształcenia – wymagania ogólne

> 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

> 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);

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

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

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):

- Użyjesz wyrażenia trójargumentowego (`a if b else c`).
- Przeanalizujesz słownikowy typ danych (`dict`).
- Scharakteryzujesz różnice między słownikiem a listą.
- Użyjesz słownika jako alternatywy dla konstrukcji wielokrotnego wyboru.
- Dokonasz analizy instrukcji `match/case`.

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 (ćwiczenia wykorzystujące instrukcję `match/case` wykonać można wyłącznie w wersji 3.10).

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wybór wielokrotny w języku Python". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematu i celów lekcji. Określenie wiążących dla uczniów kryteriów sukcesu.
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Na forum klasy uczniowie analizują przedstawione w niej przykłady i implementują je na swoich komputerach.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Schemat interaktywny". Uczniowie wspólnie zapoznają się z treścią zawartego w niej multimedium. Zapisują ewentualne problemy i pytania. Po czym następuje dyskusja, w trakcie której nauczyciel wyjaśnia niezrozumiałe treści.
3. **Ćwiczenie umiejętności.** Nauczyciel przechodzi do sekcji „Sprawdź się". Uczniowie indywidualnie rozwiązują ćwiczenie 1, nauczyciel weryfikuje poprawność wykonania zadania.
4. W kolejnym etapie uczniowie dobierają się w pary i wykonują ćwiczenia nr 2 i 3 z sekcji „Sprawdź się". Następnie konsultują swoje rozwiązania z inną parą uczniów i ustalają jedną wersję odpowiedzi.
5. Uczniowie wykonują w parach ćwiczenie 4 z sekcji „Sprawdź się".

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).

Praca domowa:

1. Uczniowie wykonują ćwiczenia 5-7 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.