

Laboratorium nr 2

Tematyka : Instrukcja warunkowa if elif else, praca z plikami

Instrukcje warunkowe to fundamentalne elementy programowania, które pozwalają na wykonanie określonych bloków kodu w zależności od spełnienia zadanych warunków. Python, jako jeden z najpopularniejszych języków programowania, oferuje prostą i intuicyjną składnię do implementacji tych instrukcji.

Do oznaczania bloków kodu w Pythonie używamy wyłącznie wcięć. Standardowym wcięciem w Pythonie są 4 spacje, chociaż tabulator i inne wcięcia działają tak długo, jak są konsekwentnie stosowane.

Instrukcja warunkowa if

if warunek: instrukcje	Blok instrukcje jest wykonany, jeżeli zostanie spełniony warunek. Jeśli jest jedna instrukcja, to ona może być zapisana w tej samej linii co if (if warunek: instrukcja)	x= 9 if x != 0: print("różna od 0")
if warunek: instrukcjeJeśliPrawda else: instrukcjeJeśliFałsz	Pełna instrukcja warunkowa. Jeżeli zostanie spełniony warunek wykonywane są instrukcjeJeśliPrawda, w przeciwnym wypadku – instrukcjeJeśliFałsz	if x%2 == 1: print("nieparzysta") else: print("parzysta")
if warunek1: instrukcje1Prawda elif warunek2: instrukcje2Prawda else: instrukcje	Instrukcja wielowarunkowa. Zostanie wykonany ten blok instrukcji, której warunek zostanie spełniony jako pierwszy.	if x > 0: print("dodatnia") elif x < 0: print("ujemna") else: print("zero")

Instrukcja if umożliwia wykonanie bloku kodu tylko wtedy, gdy określony warunek jest spełniony.

```
if <warunek>:  
    <blok kodu>
```

...

Przykład:

```
x = 100
```

```
if x > 25:  
    print("x jest większe od 25")
```

W tym przykładzie tekst komunikat "x jest większe od 25" zostanie wyświetlony, ponieważ warunek $x > 25$ jest prawdziwy.

Instrukcja **else** umożliwia wykonanie bloku kodu, gdy warunek w instrukcji **if** nie jest spełniony.

```
if <warunek>:  
    <blok kodu>  
    ...  
else:  
    <blok kodu>  
    ...
```

Przykład:

```
x = 30
```

```
if x > 50:  
    print("x jest większe od 50")  
else:  
    print("x jest mniejsze lub równe 50")
```

W tym przykładzie komunikat „x jest mniejsze lub równe 50” zostanie wyświetlony, ponieważ warunek $x > 50$ jest fałszywy

Instrukcja **elif** jest używana w połączeniu z **if** i **else** do tworzenia bardziej złożonych struktur warunkowych, nazywamy ją instrukcją wielowarunkową. Działa jako dodatkowy warunek, który jest sprawdzany tylko wtedy, gdy warunek w **if** nie jest spełniony. Umożliwia to sprawdzenie wielu warunków po kolei.

```
if <warunek>:  
    <blok kodu>  
    ...  
elif <warunek>:  
    <blok kodu>  
    ...  
elif <warunek>:  
    <blok kodu>  
    ...
```

...

else:

<blok kodu>

...

Przykład:

```
x = int(input("Podaj wartość liczby x"))
```

```
if x > 15:
```

```
    print("x jest większe od 15")
```

```
elif x > 10:
```

```
    print("x jest większe od 10, ale mniejsze lub równe 15")
```

```
elif x > 5:
```

```
    print("x jest większe od 5, ale mniejsze lub równe 10")
```

```
else:
```

```
    print("x jest mniejsze lub równe 5")
```

W takiej pracy programu były sprawdzane kolejno 3 warunki, czy $x > 15$, $x > 10$, $x > 5$. Jeżeli żaden z warunków nie jest spełniony, wykonywany jest blok kodu else.

Wynikiem pracy programu jest zwrócenie informacji „x jest większe od 5, ale mniejsze lub równe 10”

Możemy zagnieżdżać instrukcje warunkowe, tj. umieszczać jedną instrukcję warunkową wewnątrz innej. To pozwala na bardziej skomplikowane konstrukcje warunkowe.

Przykład:

```
wiek = 18
```

```
if wiek >= 18:
```

```
    print("Jesteś pełnoletni.")
```

```
if wiek >= 21:
```

```
    print("Możesz kupować alkohol w USA.")
```

True/False

W Pythonie, jak w wielu innych językach programowania, typ danych logiczny reprezentuje wartości prawdy lub fałszu. Jest to odzwierciedlenie idei binarnych – '1' dla prawdy (True) i '0' dla fałszu (False). Wartości True (prawda) odpowiada liczba całkowita 1. a wartość False (fałsz) jest

reprezentowana przez liczbę całkowitą 0. **Należy pamiętać, że Python rozpoznaje dowolną pustą strukturę danych jako fałsz, a dowolną niepustą strukturę danych jako prawdę.**

Operacje logiczne: Python obsługuje operacje logiczne, takie jak and (i), or (lub) i not (nie). Te operatory są używane w kontekście wyrażeń logicznych i warunków.

Operacje logiczne

warunek1 = True

warunek2 = False

wynik_i = warunek1 and warunek2 # "i" logiczne (and)

wynik_lub = warunek1 or warunek2 # "lub" logiczne (or)

wynik_negacji = not warunek1 # Negacja (not)

Zad 1.

Napisz prosty program, który na podstawie podanej przez Studenta liczby zdobytych punktów, poinformuje go o rezultacie egzaminu.

Każdy Student, który zdobył powyżej 80 punktów zalicza egzamin w terminie 0

Studenci którzy otrzymali liczbę punktów z przedziału 50-80, mogą poprawić jego wynik.

Studenci, którzy zdobyli poniżej 50 punktów, muszą go poprawić.

Zad. 2

Napisz program porządkowania trzech liczb x, y i z. Od najmniejszej do największej, bez użycia wbudowanych funkcji

Zad. 3

Zmienna *Nazwa_pliku* przechowującej jego nazwę.

Sprawdź, czy plik o podanej nazwie jest z rozszerzeniem '.xlsx'.

Nazwa_pliku = 'Raport_maj.xlsx'

Wydrukuj do konsoli 'Tak' jeśli to prawda, przeciwnie 'Nie'.

Podpowiedz:

Skorzystaj z metody *endswith* (jest to metodą wbudowaną dla obiektów typu *str* w Pythonie), sprawdza ona, czy ciąg znaków (*Nazwa_pliku*) kończy się podanym sufiksem (w tym przypadku '.xlsx').

Zad. 4 dodatkowe

Jesteś menagerem drużyny piłkarskiej i chcesz obliczyć łączny wynik drużyny na podstawie liczby strzelonych przez nią bramek i ewentualnie zdobytych dodatkowych punktów. Napisz program, który dokona stosownych kalkulacji po wprowadzeniu liczby goli zdobytych przez drużynę.

Utworzone są dwie zmienne `gol` i `bonus`, gdzie `gol` to liczba całkowita reprezentująca liczbę bramek zdobytych przez drużynę, a `bonus` to liczba całkowita reprezentująca wszelkie możliwe punkty bonusowe dla drużyny.

Następnie użyj instrukcji warunkowej do obliczenia całkowitego wyniku zespołu zgodnie z następującymi zasadami:

- każda zdobyta bramka to 10 punktów,
- jeśli drużyna zdobędzie więcej niż 5 bramek, zdobywa dodatkowe 5 punktów bonusowych,
- jeśli drużyna zdobędzie więcej niż 10 bramek, zdobywa dodatkowe 10 punktów bonusowych

a) Po zdobyciu 5 goli drużyna otrzymuje 5 punktów bonusowych. Jeśli drużyna zdobędzie więcej niż 10 goli, to otrzyma za nie 10 punktów bonusowych dodatkowo

Oblicz całkowity wynik drużyny, dodając punkty zdobyte ze zdobytych bramek i wszelkie stosowne punkty bonusowe. Wynik wydrukuj do konsoli.

b) Punkty bonusowe po przekroczeniu 5 i 10 punktów są sumowane, tzn. po przekroczeniu więcej niż 10 bramek drużyna zdobywa obydwa bonusy.

Oblicz całkowity wynik drużyny, dodając punkty zdobyte ze zdobytych bramek i wszelkie stosowne punkty bonusowe. Wynik wydrukuj do konsoli.

Czytanie zawartości pliku

Plik możemy otworzyć w jednym z trzech trybów:

tryb "r" — tryb do czytania pliku (read)

tryb "w" — tryb do zapisywania danych do pliku (write)

tryb "a" — tryb do dopisywania zawartości na koniec pliku (append)

Otwieranie pliku tekstowego

```
plik = open("dane.txt", "r") # otwieramy plik w trybie do odczytu (r - read)
```

```
print(plik.read()) # wypisanie zawartości pliku
```

```
plik.close() # zamknięcie pliku
```

Funkcja `open()` otwiera plik `dane.txt` w trybie do czytania. Obiekt `plik` zostaje skojarzony z plikiem `dane.txt` i od tego momentu możemy użyć metody `read()` aby pobrać zawartość pliku i np. wyświetlić ją na ekranie monitora.

Pamiętaj, że po skończonej pracy na pliku należy go zamknąć metodą `close()`, w przeciwnym razie może to prowadzić do problemów związanych z zasobami i niewłaściwym zwalnianiem pamięci.

Drugi sposób otwierania pliku w wybranym trybie

with open("dane.txt", "r") as plik:

```
    print(plik.read())
```

W powyższym przykładzie używamy bloku `with`, który automatycznie zarządza zamknięciem pliku po zakończeniu bloku. Nie musimy wywoływać metody `close()`. Blok `with` jest często używany do obsługi plików w Pythonie, ponieważ gwarantuje właściwe zarządzanie zasobami.

Czytanie pliku wierszami

with open("dane.txt", "r") as plik:

```
    for linia in plik:
```

```
        print(linia)
```

Zapisywanie danych do pliku

with open("out.txt", "w") as plik:

```
    plik.write("Ała ma kota")
```

with open("out.txt", "r") as plik:

```
    print(plik.read()) # Ała ma kota
```

zad. 5

a)

Odczytaj podany plik `notwania_gieldowe.txt` zawierający dane dotyczące notowań kilku spółek. Wydrukuj każdą linię do konsoli.

b)

Dopisz do pliku `notwania_gieldowe.txt`, w kolejnej linii dane dotyczące nowej spółki: ALR, 113. Wydrukuj każdą linię ponownie do konsoli.

Zad. 6 Napisz skrypt w Pythonie, który sprawdza, czy litera wprowadzona przez użytkownika jest duża czy mała

Zad 7. Podana jest poniższa zmienna przechowująca ciąg znaków - hasło:

```
Hasło = 'pk47!jy0893'
```

Sprawdź, czy podane hasło ma wymaganą długość 11 znaków oraz zawiera znak specjalny '!'. Jeżeli tak, wydrukuj do konsoli „Hasło jest poprawne”, w przeciwnym razie „Hasło jest nie poprawne”.

Operator wycięcia:

```
sekwencja[start:stop:step]
```

start (opcjonalny): Indeks początkowy, od którego zaczyna się wycinek. Element o tym indeksie jest włączony do wyniku. Domyślnie start wynosi 0.

stop (opcjonalny): Indeks końcowy, do którego wycinek jest pobierany, ale ten element nie jest już włączony do wyniku. Domyślnie stop to długość sekwencji.

step (opcjonalny): Krok, z jakim pobierane są elementy. Domyślnie wynosi 1, co oznacza, że elementy są pobierane kolejno. Ustawienie go na -1 pozwala odwrócić sekwencję.

Wycinanie części listy

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Wyciągnięcie elementów od indeksu 2 do 5 (nie włącznie)

```
podlista = lista[2:5]
```

```
print(podlista) # Wynik: [2, 3, 4]
```

Pominięcie indeksu 'stop'

```
podlista = lista[5:] # Od 5-tego indeksu do końca
```

```
print(podlista) # Wynik: [5, 6, 7, 8, 9]
```

Ostatnie trzy elementy

```
podlista = lista[-3:]
```

```
print(podlista) # Wynik: [7, 8, 9]
```

Odwrócenie listy za pomocą ujemnego kroku

```
podlista = lista[::-1]
```

```
print(podlista) # Wynik: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Co drugi element z zakresu od 1 do 8

```
podlista = lista[1:8:2]
```

```
print(podlista) # Wynik: [1, 3, 5, 7]
```

Zad 8.

Stwórz program, który wykorzystując operator wycinania z podanego ciąg znaków (zmienna text) wyodrębni:

pierwsze trzy znaki

ostatnie dwa znaki

```
text = 'Studiuje-Informatykę'
```

Wynik wydrukuj do konsoli.

Zad 9. dodatkowe

Napisz skrypt zmieniający wszystkie duże litery małe i na odwrót.

Podpowiedź: skorzystaj z metody swapcase().