- Laboratorium Wstępu do Programowania: 2
- Temat: Instrukcja warunkowa if elif else, praca z plikami
- Patryk Paściak
- Numer albumu: 73154
- Grupa: IN. INP L6
- Prowadzący: mgr inż. Przemysław Skubel

Zadanie 1

Cel zadania

- Napisanie prostego programu warunkowego, który analizuje liczbę punktów uzyskanych przez studenta.
- Zapoznanie się z instrukcjami warunkowymi if, elif, else.
- Wyświetlanie odpowiedniego komunikatu w zależności od wyniku.

Przebieg zadania

1. Program pobiera od użytkownika liczbę zdobytych punktów (punkty).
2. Za pomocą instrukcji warunkowych sprawdza, do którego przedziału należy wartość:
    a. powyżej 80 pkt – egzamin zaliczony w terminie 0,
    b. od 50 do 80 pkt – możliwość poprawy wyniku,
    c. poniżej 50 pkt – konieczność poprawy egzaminu.
3. Na podstawie wyniku wyświetlany jest odpowiedni komunikat w konsoli.

Kod Programu

```python
punkty = float(input("Podaj liczbę zdobytych punktów: "))

if punkty > 80:
    print("Zaliczyłeś egzamin w terminie 0. Gratulacje!")
elif 50 <= punkty <= 80:
    print("Możesz poprawić wynik egzaminu.")
else:
    print("Musisz poprawić egzamin.")

```

Wynik Programu

```
Podaj liczbę zdobytych punktów: 11
Musisz poprawić egzamin.
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
Podaj liczbę zdobytych punktów: 80
Możesz poprawić wynik egzaminu.
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
Podaj liczbę zdobytych punktów: 81
Zaliczyłeś egzamin w terminie 0. Gratulacje!
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
```

Analiza programu

- Program wykorzystuje instrukcję warunkową if-elif-else do porównywania wartości liczbowych.
- Operator logiczny <= zapewnia, że graniczne wartości (np. 50 i 80) są poprawnie przypisane do odpowiednich przypadków.
- Funkcja input() pobiera dane z klawiatury, a float() umożliwia wprowadzenie liczb z miejscami po przecinku.
- Program działa poprawnie dla każdej liczby punktów i daje intuicyjny wynik.

Wnioski

- Instrukcje warunkowe są podstawowym narzędziem do podejmowania decyzji w programie.
- Poprawne ustawienie zakresów warunków gwarantuje właściwą interpretację danych.
- Program w prosty sposób ilustruje praktyczne zastosowanie logiki w ocenie wyników studenta.

Zadanie 2

Cel zadania

- Napisanie programu, który porządkuje trzy liczby od najmniejszej do największej.
- Ćwiczenie instrukcji warunkowych if oraz operacji zamiany wartości między zmiennymi.
- Zrozumienie działania prostego algorytmu sortowania przez porównania.

Przebieg zadania

1. Użytkownik wprowadza trzy liczby (x, y, z).
2. Program porównuje liczby parami i zamienia ich wartości miejscami, jeśli są w złej kolejności.
3. Kolejne porównania i zamiany gwarantują, że po wykonaniu wszystkich instrukcji liczby są uporządkowane rosnąco.
4. Program wyświetla wynik w formacie: od najmniejszej do największej.

Kod Programu

```
x = float(input("Podaj pierwszą liczbę (x): "))
y = float(input("Podaj drugą liczbę (y): "))
z = float(input("Podaj trzecią liczbę (z): "))

if x > y:
    x, y = y, x
if y > z:
    y, z = z, y
if x > y:
    x, y = y, x

print("Liczby od najmniejszej do największej:", x, y, z)
```

Wynik programu

```
Podaj pierwszą liczbę (x): 4
Podaj drugą liczbę (y): 5
Podaj trzecią liczbę (z): 6
Liczby od najmniejszej do największej: 4.0 5.0 6.0
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
```

Analiza programu

- **P**rogram wykorzystuje trzy porównania i ewentualne zamiany miejsc wartości przy użyciu składni x, y = y, x.
- Kolejność porównań sprawia, że po każdym kroku największa liczba przesuwa się na koniec, a najmniejsza na początek.
- Dzięki trzem instrukcjom if program jest prosty, a jednocześnie skuteczny — zawsze uzyskuje poprawne uporządkowanie.
- Nie korzystano z żadnych funkcji takich jak sorted() czy min(), co spełnia warunki zadania.

Wnioski

- Program pokazuje, że nawet bez wbudowanych funkcji można napisać prosty algorytm sortujący.
- Instrukcje warunkowe i zamiana miejsc wartości są kluczowymi elementami w podstawowych metodach sortowania.
- To podejście ilustruje zasadę działania prostych algorytmów porządkujących, takich jak sortowanie bąbelkowe.

Zadanie 3

Cel zadania

- Poznanie metody endswith() służącej do sprawdzania, czy łańcuch znaków kończy się określonym sufiksem.
- Napisanie programu, który sprawdza, czy plik ma rozszerzenie .xlsx.
- Ćwiczenie instrukcji warunkowych i operacji na ciągach znaków w Pythonie.

Przebieg zadania

1. Do zmiennej Nazwa_pliku przypisana zostaje nazwa pliku.
2. Program sprawdza, czy ciąg znaków kończy się rozszerzeniem .xlsx przy pomocy metody endswith().

3. Jeśli warunek jest spełniony, program wypisuje „TAK”, w przeciwnym wypadku „NIE”.

Kod Programu

```
Nazwa_pliku= 'Raport_maj.xlsx'
if Nazwa_pliku.endswith(".xlsx"):
    print("TAK")
else:
    print("NIE")
```

Wynik Programu

```
TAK
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
```

Analiza programu

- Metoda endswith() zwraca wartość logiczną True lub False.
- Jeśli wynik to True, instrukcja if wykonuje pierwszy blok (print("TAK")), w przeciwnym razie — blok else.
- Program działa poprawnie niezależnie od długości i zawartości nazwy pliku.
- To prosty przykład sprawdzania rozszerzenia plików w Pythonie.

Wnioski

- Metoda endswith() jest bardzo użyteczna przy pracy z nazwami plików i ich rozszerzeniami.
- Instrukcje warunkowe umożliwiają szybkie podejmowanie decyzji na podstawie wyniku logicznego.
- Program jest prosty, ale ilustruje praktyczne zastosowanie metod dla typu str.

Zadanie 4 - A

Cel zadania

- Stworzenie programu, który oblicza całkowity wynik drużyny piłkarskiej w oparciu o zdobyte gole oraz punkty bonusowe.
- Ćwiczenie wykorzystania instrukcji warunkowych if, elif, else.
- Nauka logicznego podejścia do warunków zależnych od liczby bramek.

Wariant A – Bonus zastępujący poprzedni

Przebieg zadania

1. Użytkownik podaje:
    a. liczbę zdobytych bramek (gol),
    b. dodatkowe punkty bonusowe (bonus).
2. Każda bramka warta jest 10 punktów.
3. Jeśli drużyna zdobyła:
    a. więcej niż 5 bramek – otrzymuje 5 punktów bonusowych,
    b. więcej niż 10 bramek – otrzymuje 10 punktów bonusowych (zastępuje poprzedni bonus).
4. Program oblicza łączny wynik i wyświetla szczegółowe dane.

Kod Programu

```python
gol = int(input("Podaj liczbę zdobytych bramek: "))
bonus = int(input("Podaj dodatkowe punkty bonusowe: "))

punkty = gol * 10

if gol > 10:
    punkty_bonusowe = 10
elif gol > 5:
    punkty_bonusowe = 5
else:
    punkty_bonusowe = 0

wynik = punkty + punkty_bonusowe + bonus

print(f"Punkty za bramki: {punkty}")
print(f"Punkty bonusowe: {punkty_bonusowe}")
print(f"Wynik końcowy: {wynik}")
```

Wynik Programu

```
Podaj liczbę zdobytych bramek: 6
Podaj dodatkowe punkty bonusowe: 0
Punkty za bramki: 60
Punkty bonusowe: 5
Wynik końcowy: 65
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
Podaj liczbę zdobytych bramek: 11
Podaj dodatkowe punkty bonusowe: 0
Punkty za bramki: 110
Punkty bonusowe: 10
Wynik końcowy: 120
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
```

Analiza programu

- Każdy gol daje 10 punktów.
- Instrukcja if-elif-else zapewnia, że tylko jeden bonus zostanie przyznany — nie są sumowane.
- Zmienna bonus pozwala użytkownikowi wprowadzić ewentualne dodatkowe punkty (np. za styl gry).
- Program jest prosty, czytelny i łatwy do rozbudowy.

Wnioski

- Instrukcje warunkowe pozwalają kontrolować logikę programu w zależności od wartości wejściowych.
- Algorytm poprawnie wylicza wynik końcowy w różnych przypadkach punktacji.
- Kod można łatwo modyfikować, np. zmieniając wartości punktowe lub zasady bonusów.

Zadanie 4 – B

Wariant B – Bonusy sumowane (po przekroczeniu 10 goli drużyna dostaje oba bonusy)

Przebieg zadania

1. Program działa podobnie jak w wariancie A.
2. Zasady bonusów:
    a. powyżej 5 bramek – 5 punktów,

b. powyżej 10 bramek – 10 punktów dodatkowych,

c. razem 15 punktów bonusowych przy liczbie goli > 10.

3. Wynik końcowy to suma punktów z bramek, bonusów i ewentualnych punktów dodatkowych od użytkownika.

Kod Programu

```python
gol = int(input("Podaj liczbę zdobytych bramek: "))
bonus = int(input("Podaj dodatkowe punkty bonusowe: "))

punkty = gol * 10

if gol > 10:
    punkty_bonusowe = 15
elif gol > 5:
    punkty_bonusowe = 5
else:
    punkty_bonusowe = 0


wynik = punkty + punkty_bonusowe + bonus

print(f"Punkty za bramki: {punkty}")
print(f"Punkty bonusowe: {punkty_bonusowe}")
print(f"Wynik końcowy: {wynik}")
```

Wynik Programu

```
Podaj liczbę zdobytych bramek: 6
Podaj dodatkowe punkty bonusowe: 0
Punkty za bramki: 60
Punkty bonusowe: 5
Wynik końcowy: 65
PS C:\Users\patry\Desktop\Lab.Progra
Podaj liczbę zdobytych bramek: 11
Podaj dodatkowe punkty bonusowe: 0
Punkty za bramki: 110
Punkty bonusowe: 15
Wynik końcowy: 125
PS C:\Users\patry\Desktop\Lab.Progra
```

Analiza programu

- Program różni się od poprzedniego tylko sposobem naliczania bonusów — tutaj są one sumowane.
- Dzięki temu za więcej niż 10 goli drużyna dostaje zarówno 5, jak i 10 punktów (razem 15).
- Kod zachowuje prostotę i czytelność, a jednocześnie pokazuje, jak można w prosty sposób modyfikować logikę programu.

Wnioski

- Warunki w Pythonie można łatwo dostosować do różnych zasad gry.
- Nawet drobna zmiana w logice instrukcji if może istotnie wpłynąć na końcowy wynik programu.
- Program dobrze obrazuje praktyczne zastosowanie instrukcji warunkowych w prostych kalkulatorach punktowych.

Zadanie 5A

Cel zadania

- Nauka odczytywania danych z pliku tekstowego.
- Utrwalenie wykorzystania instrukcji with open() oraz iteracji po liniach pliku.
- Zapoznanie się z metodą strip().

Przebieg zadania

1. Otwieramy plik notowania_gieldowe.txt w trybie odczytu "r" z kodowaniem UTF-8.
2. Iterujemy po każdej linii za pomocą pętli for.
3. Każda linia jest wypisywana na ekran z pomocą funkcji print().
4. Metoda .strip() usuwa znaki końca linii (\n), aby wynik wyglądał czytelnie.

Kod Programu

```
with open("notowania_gieldowe.txt", "r", encoding="utf-8") as plik:
    for linia in plik:
        print(linia.strip())
```

Wynik Programu

```
KGHM, 123
Tauron, 150
Orange, 45
PGE, 24
PKN Orlen, 70
PKO BP, 56
```

Analiza programu

- Konstrukcja with open(...) as plik: automatycznie zamyka plik po zakończeniu pracy.
- Iteracja po pliku jest najprostszym i najwydajniejszym sposobem czytania wielu linii.

- Metoda .strip() poprawia estetykę wyświetlania usuwając zbędne białe znaki.

Wnioski

- Program poprawnie odczytuje plik i drukuje jego zawartość.
- Użycie with oraz .strip() jest dobrą praktyką programistyczną.
- Kod jest prosty, przejrzysty i w pełni zgodny z poleceniem.

Zadanie 5A

Cel zadania

- Nauka dopisywania danych do istniejącego pliku.
- Utrwalenie trybu otwierania plików: "a" (append).
- Ponowny odczyt całego pliku i wyświetlenie wyników w konsoli.

Przebieg zadania

1. Plik notowania_gieldowe.txt został otwarty w trybie dopisywania "a".
2. Do pliku dopisano nową linię zawierającą dane spółki:
   ALR;113
3. Ponownie otwarto plik w trybie odczytu "r".
4. Wszystkie linie zostały wydrukowane w konsoli, z usunięciem znaków przejścia do nowej linii dzięki metodzie .strip().

Kod Programu

```
with open("notowania_gieldowe.txt", "a", encoding="utf-8") as plik:
    plik.write("ALR, 113\n")

with open("notowania_gieldowe.txt", "r", encoding="utf-8") as plik:
    for linia in plik:
        print(linia.strip())
```

Wynik Programu

```
KGHM, 123
Tauron, 150
Orange, 45
PGE, 24
PKN Orlen, 70
PKO BP, 56
ALR, 113
```

Analiza

- Tryb "a" zapewnia, że nowe dane dopisywane są na końcu pliku, nie nadpisując istniejącej treści.
- Po dopisaniu danych konieczne jest ponowne otwarcie pliku, aby wyświetlić jego aktualną zawartość.
- strip() usuwa znaki \n, dzięki czemu linie wyświetlają się estetycznie.

Wnioski

- Program działa poprawnie i spełnia wymagania zadania.
- Użycie with open() zapewnia automatyczne zamykanie pliku.
- Tryb "a" jest idealny do uzupełniania danych w pliku tekstowym.

Zadanie 6

Cel zadania

- Napisanie programu, który sprawdza, czy wprowadzony znak jest dużą czy małą literą.
- Ćwiczenie instrukcji warunkowych if, elif, else.
- Poznanie metod wbudowanych dla typu str: isupper() oraz islower().

Przebieg zadania

1. **Użytkownik wprowadza znak (literę) z klawiatury.**
2. **Program najpierw sprawdza, czy użytkownik wpisał dokładnie jeden znak.**
3. **Następnie weryfikuje:**
    a. **czy litera jest duża (isupper()),**
    b. **czy litera jest mała (islower()).**
4. **Jeśli znak nie jest literą (np. cyfra lub symbol), program informuje użytkownika o błędzie**

Kod Programu

```python
litera = input("Podaj jedną literę: ")

if len(litera) != 1:
    print("Podaj dokładnie jedną literę.")
elif litera.isupper():
    print("Litera jest DUŻA.")
elif litera.islower():
    print("Litera jest mała.")
else:
    print("To nie jest litera.")
```

Wynik Programu

```
Podaj jedną literę: O
Litera jest DUŻA.
PS C:\Users\patry\Desktop\Lab
Podaj jedną literę: o
Litera jest mała.
PS C:\Users\patry\Desktop\Lab
Podaj jedną literę: 1
To nie jest litera.
PS C:\Users\patry\Desktop\Lab
Podaj jedną literę: ab
Podaj dokładnie jedną literę.
PS C:\Users\patry\Desktop\Lab
```

Analiza programu

- Funkcja len() sprawdza długość wprowadzonego tekstu, dzięki czemu program wymaga podania tylko jednego znaku.
- Metody isupper() i islower() działają tylko na literach, zwracając odpowiednio True lub False.
- Warunki są ułożone w logicznej kolejności, dzięki czemu program działa poprawnie nawet przy błędnym wejściu.
- Dzięki blokowi else program obsługuje przypadki, gdy znak nie jest literą (np. @, 7, !).

Wnioski

- Program skutecznie rozpoznaje, czy wprowadzony znak to mała, czy duża litera.
- Instrukcje warunkowe i metody wbudowane w typ str są bardzo przydatne do walidacji danych tekstowych.
- Sprawdzanie długości wejścia chroni program przed błędami użytkownika.

Zadanie 7

Cel zadania

- Napisanie programu, który sprawdza, czy podane hasło spełnia określone wymagania.
- Utrwalenie użycia funkcji len() oraz funkcji any() w połączeniu z pętlą warunkową.
- Ćwiczenie operatorów logicznych and.

Przebieg zadania

1. Utworzono zmienną hasło, przechowującą ciąg znaków 'pk47!jy0893'.
2. Zmienna znak_specjalny zawiera znak '!'.
3. Program sprawdza dwa warunki jednocześnie:
    a. Czy długość hasła wynosi dokładnie **11 znaków**.
    b. Czy w haśle znajduje się znak specjalny '!'.
4. Jeśli oba warunki są spełnione, program wypisuje komunikat „Hasło jest poprawne.”, w przeciwnym wypadku – „Hasło jest niepoprawne.”

Kod Programu

```
hasło = 'pk47!jy0893'
znak_specjalny = "!"
if len(hasło) == 11 and any(znak in znak_specjalny for znak in hasło):
    print("Hasło jest poprawne.")
else:
    print("Hasło jest niepoprawne.")
```

Wynik Programu

```
Hasło jest poprawne.
PS C:\Users\patry\Desktop\Lab.Programowanie\lab.2>
```

Analiza programu

- Funkcja len(hasło) sprawdza długość łańcucha znaków.
- Funkcja any() w połączeniu z wyrażeniem znak in znak_specjalny for znak in hasło sprawdza, czy w haśle znajduje się znak '!'.
- Warunek logiczny and zapewnia, że oba kryteria muszą być spełnione jednocześnie.
- Program jest prosty, czytelny i skuteczny — sprawdza wszystkie wymagane elementy hasła.

Wnioski

- Program prawidłowo rozpoznaje poprawne i niepoprawne hasła.
- Zastosowanie funkcji any() upraszcza sprawdzanie obecności określonych znaków w tekście.
- Tego typu logikę można łatwo rozbudować np. o sprawdzanie wielkich liter, cyfr lub minimalnej długości hasła.

Zadanie 8

Cel zadania

- Poznanie operatora wycinania (slice) w Pythonie.
- Ćwiczenie manipulacji łańcuchami znaków.
- Wyodrębnianie konkretnych fragmentów tekstu z podanego ciągu.

Przebieg zadania

1. Utworzono zmienną text z ciągiem znaków 'Studiuje-Informatykę'.
2. Program wyodrębnia:
    a. pierwsze trzy znaki przy użyciu text[:3],
    b. ostatnie dwa znaki przy użyciu text[-2:].
3. Wyniki wyświetlane są w konsoli przy pomocy funkcji print().

Kod Programu

```
text = 'Studiuje-Informatykę'

pierwsze_trzy = text[:3]

ostatnie_dwa = text[-2:]

print("Pierwsze trzy znaki:", pierwsze_trzy)
print("Ostatnie dwa znaki:", ostatnie_dwa)
```

Wynik Programu

```
Pierwsze trzy znaki: Stu
Ostatnie dwa znaki: kę
```

Analiza programu

- Operator wycinania [:3] pobiera znaki od początku ciągu do indeksu 3 (nie wliczając go).
- Operator [-2:] pobiera ostatnie dwa znaki ciągu.
- Program jest prosty, czytelny i skuteczny w wyodrębnianiu fragmentów tekstu.

Wnioski

- Operator wycinania (slice) jest bardzo użyteczny przy manipulacji tekstem w Pythonie.
- Pozwala łatwo wyciągać fragmenty łańcuchów znaków bez użycia dodatkowych funkcji.
- Tę technikę można stosować np. do analizy danych tekstowych, numerów identyfikacyjnych czy skracania wyrazów.

Zadanie 9

Cel zadania

- Poznanie metody swapcase() dla typu str.
- Umiejętność zamiany małych liter na duże i dużych na małe w podanym tekście.
- Utrwalenie pracy z danymi wejściowymi i funkcją print().

Przebieg zadania

1. Program pobiera ciąg znaków od użytkownika.
2. Metoda swapcase() tworzy nowy tekst, w którym:
    a. wszystkie litery małe → duże,
    b. wszystkie litery duże → małe.
3. Program wyświetla zmieniony tekst.

Kod Programu

```
text = input("Podaj tekst: ")

nowy_text = text.swapcase()

print("Tekst po zamianie liter:")
print(nowy_text)
```

Wynik Programu

```
Podaj tekst: Kocham MAteMAtyKĘ
Tekst po zamianie liter:
kOCHAM maTEmaTYkę
```

Analiza programu

- swapcase() to metoda działająca na łańcuchach znaków, automatycznie zmieniająca wielkość liter.
- Program jest krótki, czytelny i wykorzystuje najprostszy możliwy mechanizm realizujący wymagania.
- Nie trzeba stosować pętli ani ręcznego sprawdzania liter — Python robi to za nas.

Wnioski

- Metoda swapcase() pozwala bardzo łatwo manipulować wielkością liter w tekście.
- Program działa poprawnie dla każdego wprowadzonego ciągu znaków.
- Jest to wygodne narzędzie w zadaniach związanych z przetwarzaniem tekstu i formatowaniem danych.