Programy, które myślą – Instrukcje Warunkowe i Pętle

Agenda:

- Krótkie przypomnienie: Co już potrafimy? (Programy liniowe).
- Jak programy podejmują decyzje? (Instrukcja if-elif-else).
- Ćwiczenie 1: Nasz pierwszy inteligentny program.
- Łączenie warunków (and, or, not).
- Ćwiczenie 2: Program, który powtarza zadania (Pętla while).
- Zadania do samodzielnej pracy.

Przypomnienie: Co już potrafimy?

Nasze dotychczasowe programy:

- Wykonywały się liniowo (instrukcja po instrukcji).
- Potrafiły prowadzić dialog (**print**, **input**).
- Zapamiętywały dane (zmienne).
- Rozróżniały podstawowe typy danych (**str**, **int**).

Narzędzia, których dziś użyjemy:

- Operatory porównania: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- Konwersja typów (rzutowanie): **int()**.
- Słowo kluczowe: **if**.

Cel na dziś: Przejść od programów liniowych do programów, które reagują na dane.

Ćwiczenie 1: Projektujemy decyzję

Problem: Narysuj schemat blokowy dla programu, który pyta użytkownika o wiek i wyświetla specjalny komunikat **tylko wtedy**, gdy użytkownik jest pełnoletni.

Nowy blok w schemacie:

- **Romb (Decyzja)**: Zadaje pytanie, na które można odpowiedzieć "Tak" lub "Nie". Zawsze wychodzą z niego **dwie** ścieżki.

Ćwiczenie 1: Implementacja instrukcji if

Cel: Przełożenie projektu (schematu i pseudokodu) na działający kod z instrukcją **if**.

Problem: Napisz program, który pyta użytkownika o wiek i wyświetla specjalny komunikat **tylko wtedy**, gdy użytkownik jest pełnoletni. Na końcu program zawsze powinien podziękować za skorzystanie z usługi.

Ćwiczenie 2: Projektujemy wiele ścieżek

Zaprojektuj algorytm kategoryzacji wieku.

Problem: Rozbudujmy nasz program. Na podstawie podanego wieku, przypisz użytkownika do jednej z czterech kategorii i wyświetl odpowiedni komunikat:

- **dziecko (poniżej 13 lat),**
- **nastolatek (13-17),**
- **dorosły (18-64),**
- **senior (65+).**

Ćwiczenie 2: Implementacja wielu ścieżek (if-elif-else)

Zaimplementuj algorytm kategoryzacji wieku.

Problem: Rozbudujmy nasz program. Na podstawie podanego wieku, przypisz użytkownika do jednej z czterech kategorii i wyświetl odpowiedni komunikat:

- **dziecko (poniżej 13 lat),**
- **nastolatek (13-17),**
- **dorosły (18-64),**
- **senior (65+).**

Narzędzia:

- **if** - dla pierwszego warunku.
- **elif - (else if)** dla każdego kolejnego warunku.
- **else - dla przypadku, gdy żaden warunek nie jest spełniony.**

Ćwiczenie 3: Wystawianie Oceny

Cel: Samodzielne użycie łańcucha if-elif-else do rozwiązania praktycznego problemu.

Problem: Napisz program, który:

1. Pyta użytkownika o liczbę punktów z kolokwium (0-100).
2. Na podstawie punktów, wystawia i drukuje odpowiednią ocenę, zgodnie z poniższą skalą:
    - **0-50** pkt: ocena **2.0**
    - **51-60** pkt: ocena **3.0**
    - **61-70** pkt: ocena **3.5**
    - **71-80** pkt: ocena **4.0**
    - **81-90** pkt: ocena **4.5**
    - **91-100** pkt: ocena **5.0**

Wskazówka: Pomyśl o kolejności sprawdzania warunków. Czy łatwiej jest sprawdzać od najniższej, czy od najwyższej wartości?

Omówienie Ćwiczenia 3: Wystawianie Oceny

Cel: Analiza poprawnego rozwiązania i logiki warunków.

Kluczowa koncepcja: Każdy **elif** jest sprawdzany tylko, gdy poprzednie warunki były fałszywe. To pozwala na uproszczenie logiki.

Łączenie warunków: and, or, not

Cel: Tworzenie bardziej złożonych, wieloczęściowych warunków.

and (Koniunkcja - "i"):

- **warunek1 and warunek2** jest **True** tylko wtedy, gdy oba warunki są **True.**
- **Przykład:** if wiek >= 18 and ma_prawo_jazdy:

or (Alternatywa - "lub"):

- **warunek1 or warunek2** jest **True,** gdy przynajmniej jeden z warunków jest **True.**
- **Przykład:**
  if dzien == "sobota" or dzien == "niedziela":

not (Negacja - "nie"):

- **not warunek** odwraca wartość logiczną **(True -> False, False -> True).**
- **Przykład:** if not jest_zalogowany:

Ćwiczenie 4: Logika w praktyce

Cel: Samodzielne zbudowanie złożonego warunku z użyciem and i or.

Problem: "Wstęp na kolejkę górską". Napisz program, który decyduje, czy dana osoba może wejść na kolejkę.

Zasady:

1. Osoba musi mieć **co najmniej 140 cm** wzrostu.
2. **ORAZ** musi spełniać **jeden** z poniższych warunków:
    - Mieć ukończone 18 lat.
    - Mieć zgodę opiekuna.

Kroki:

- Pobierz od użytkownika wiek, wzrost i informację o zgodzie opiekuna (np. "tak" / "nie").
- Zbuduj jeden, złożony warunek **if**, który sprawdzi wszystkie zasady naraz.
- Wyświetl komunikat "Możesz wejść!" lub "Niestety, nie spełniasz warunków."
- Wskazówka: Użyj nawiasów **()**, aby jasno zgrupować warunki połączone operatorem **or**.

Omówienie Ćwiczenie 4: Logika w praktyce

Cel: Cel: Analiza złożonego warunku i roli nawiasów.

Powtarzanie zadań: Pętla ‘while’

Cel: Wykonuj blok kodu tak długo, jak pewien warunek jest prawdziwy.

Analogia: if, który sam się powtarza.

Problem: Zaprojektuj program, który odlicza od 5 do 1, a następnie wyświetla "Start!".

Uwaga na nieskończone pętle!
Wewnątrz pętli coś musi wpływać na warunek!

Pseudokod:

```
START
    USTAW licznik = 5
    DOPÓKI licznik > 0 WYKONUJ:
        WYŚWIETL licznik
        USTAW licznik = licznik - 1
    WYŚWIETL "Start!"
STOP
```

Ćwiczenie 5: Implementacja pętli ‘while’

Cel: Przełożenie projektu odliczania na działający kod z pętlą while.

Kluczowe elementy składni:

- **while warunek**: - linia rozpoczynająca pętlę, zakończona dwukropkiem.
- **Wcięty blok kodu** - wykonuje się w każdej iteracji pętli.
- **licznik -= 1** - skrócony zapis operacji. Działa też dla `+=`, `*=`, `/=`.

Ćwiczenie 6: Interaktywna pętla

Cel: Stworzenie programu, który działa w pętli i kończy się na komendę użytkownika.

Problem: Napisz program, który działa jak "magiczna papuga".

- **Program w nieskończonej pętli prosi użytkownika o wpisanie czegoś.**
- **Program powtarza dokładnie to, co wpisał użytkownik.**
- **Pętla kończy się, gdy użytkownik wpisze słowo "koniec".**

Nowe narzędzia:

- **while True:** - Jak stworzyć pętlę, która sama z siebie się nie kończy?
- **break** - Jak z takiej pętli "awaryjnie" wyskoczyć?

Omówienie Ćwiczenia 6

Cel: Analiza rozwiązania i utrwalenie wzorca pętli sterowanej przez użytkownika.

Kluczowe elementy wzorca while-if-break:

- **while True:** - Tworzy pętlę, która sama z siebie się nie zakończy.
- **if warunek_wyjscia:** - Wewnątrz pętli umieszczamy "strażnika", który sprawdza, czy nadszedł czas na zakończenie.
- **break** - "Wyjście awaryjne", które natychmiast kończy pętlę.

Ćwiczenie 7: Sterowanie pętlą – instrukcja ‘continue’

Cel: Zrozumienie i zastosowanie instrukcji continue do pomijania iteracji.

Zaimplementuj program: "Procesor Komend"

1. Napisz program, który w pętli **while True** prosi użytkownika o komendę.
2. Jeśli komenda to "**koniec",** program kończy działanie (użyj **break**).
3. Jeśli komenda zaczyna się od znaku # (jest to komentarz), program powinien ją zignorować i od razu poprosić o następną komendę (użyj **continue**).
4. W każdym innym przypadku, program powinien "wykonać" komendę, czyli wyświetlić np. **Wykonuję: [komenda] .**

Nowe narzędzia:

- **continue** - przerywa bieżący obieg pętli i przechodzi do następnego.
- **tekst.startswith(prefix)** - metoda stringów, która sprawdza, czy tekst zaczyna się od danego prefiksu.

Omówienie Ćwiczenia 7: “Procesor komend”

Cel: Analiza rozwiązania i utrwalenie różnicy między break a continue.

Podsumowanie sterowania pętlą:

- **break: Wyjście awaryjne.** Całkowicie kończy pętlę.
- **continue: Pomiń i leć dalej.** Kończy bieżącą iterację i przechodzi do następnej.

Pętla for: “Dla każdego elementu w kolekcji”

Cel: Wykonaj blok kodu dla każdego elementu w sekwencji (np. w tekście).

while vs for:

- **while:** "Rób, dopóki warunek jest prawdziwy". Idealna, gdy nie znamy liczby powtórzeń.
- **for:** "Zrób to dla każdego elementu z tej kolekcji". Idealna, gdy mamy zbiór do przetworzenia.

```
for zmienna_tymczasowa in sekwencja:
    # Blok kodu wykonywany dla każdego elementu.
    # 'zmienna_tymczasowa' przechowuje aktualny element.
```

Ćwiczenie 8: Nasza pierwsza pętla ‘for’

Cel: Praktyczne zastosowanie pętli **for** do analizy tekstu.

Problem: "Licznik samogłosek"

- Poproś użytkownika o podanie dowolnego słowa.
- Stwórz zmienną licznik_samoglosek i ustaw ją na **0**.
- Użyj pętli **for**, aby przejść przez każdą literę w podanym słowie.
- Wewnątrz pętli, użyj instrukcji if, aby sprawdzić, czy litera jest samogłoską (**a, e, i, o, u, y**).
- Jeśli tak, zwiększ **licznik_samoglosek** o 1.
- Po zakończeniu pętli, wyświetl wynik.

Wskazówka: Aby program działał dla "Anna" i "anna" tak samo, użyj metody .lower() na słowie od użytkownika.

Zadanie: Zaimplementuj program "Licznik samogłosek".

Omówienie Ćwiczenia 8 ("Licznik samogłosek")

Cel: Analiza rozwiązania i utrwalenie kluczowych wzorców.

Kluczowe wzorce i techniki:

- **Wzorzec Akumulatora**: Inicjalizacja zmiennej (**0**) przed pętlą i aktualizowanie jej wewnątrz.
- **Operator in:** Czytelny i wydajny sposób na sprawdzenie, czy element znajduje się w kolekcji.
- **Stała SAMOGLOSKI**: Dobra praktyka - zamiast "magicznego" tekstu w kodzie, używamy nazwanej zmiennej.

Pętla ‘for’ z range(): powtarzanie zadań

Problem: Jak powtórzyć jakąś czynność dokładnie N razy?

Narzędzie: range() - generuje sekwencję liczb.

range(stop) -> od 0 do stop-1.

- **range(5) -> 0, 1, 2, 3, 4**

range(start, stop) -> od start do stop-1.

- **range(2, 5) -> 2, 3, 4**

range(start, stop, step) -> od start do stop-1, co step.

- **range(0, 10, 2) -> 0, 2, 4, 6, 8**

Ćwiczenie 7: Proste powtórzenia

- **Cel:** Użycie pętli **for** z **range()** do wykonania zadania N razy.
- **Zadanie:** Napisz program, który 5 razy wyświetli komunikat "To jest powtórzenie numer: [numer]".

Ćwiczenie 10: Pętla ‘for’ z warunkiem ‘if’ (”FizzBuzz”)

Cel: Połączenie pętli **for** z logiką **if-elif-else** do rozwiązania klasycznego problemu.

Problem: "FizzBuzz"

a) Napisz program, który iteruje przez liczby od 1 do 100 (włącznie).

b) Jeśli liczba jest podzielna przez **3**, zamiast liczby wyświetl słowo **"Fizz".**

c) Jeśli liczba jest podzielna przez **5**, zamiast liczby wyświetl słowo **"Buzz".**

d) Jeśli liczba jest podzielna **jednocześnie przez 3 i 5**, wyświetl **"FizzBuzz"**.

e) W każdym innym przypadku, po prostu wyświetl liczbę.

Nowe narzędzie: Operator Modulo %

- **a % b** zwraca **resztę z dzielenia a przez b.**
- **10 % 3 -> 1**
- **12 % 3 -> 0 (oznacza, że 12 jest podzielne przez 3)**

Zadanie: Zaimplementuj program "FizzBuzz".

Wskazówka: Kolejność sprawdzania warunków ma ogromne znaczenie!

Omówienie Ćwiczenie 10 (“FizzBuzz”)

Cel: Analiza rozwiązania i utrwalenie kluczowej roli kolejności warunków.

Kluczowa lekcja: W łańcuchu **if-elif-else**, kolejność ma fundamentalne znaczenie. Zawsze zaczynaj od najbardziej specyficznego warunku.

Podsumowanie: Od Zmiennych do Struktur Danych

Co dzisiaj zrobiliśmy?

- Daliśmy naszym programom **inteligencję** (**if-elif-else**).
- Nauczyliśmy je **cierpliwości** i powtarzania zadań (**while**, **for**).
- Zyskaliśmy **precyzyjną kontrolę** nad ich działaniem (**break**, **continue**).

Kluczowa lekcja: Opanowaliście Państwo **układ nerwowy** każdego programu. Potraficie już sterować jego zachowaniem w zależności od danych i zdarzeń.

Na następnych laboratoriach:

- Zaczniemy budować szkielet dla naszych danych.
- Poznamy struktury danych: listy, krotki i zbiory.
- Nauczymy się przechowywać i przetwarzać wiele informacji naraz.