Sprawozdanie z Laboratorium nr 1

Imię i nazwisko: Oskar Wójs

Nr indeksu: 73167

Data: 24.10.2025

Zadanie 1

Cel zadania: Celem zadania było zrozumienie zasad działania podstawowych operatorów arytmetycznych w języku Python
oraz poznanie mechanizmu dynamicznego typowania danych. Celem dodatkowym było rozróżnienie między działaniami
na liczbach całkowitych (int) i zmiennoprzecinkowych (float) oraz zapoznanie się z różnicami w wynikach tych operacji.

Przebieg zadania: W interpreterze Pythona wykonano różne działania matematyczne, sprawdzając ich wyniki i typy danych.
Zastosowano funkcję type() w celu identyfikacji rodzaju danych zwracanego przez daną operację. Przeanalizowano zachowanie
operatorów: +, -, *, /, //, %, ** oraz zasady konwersji automatycznej typów w Pythonie.

Kod programu:
1. 1 + 2
2. 1 + 4.5
3. 3 / 2
4. 4 / 2
5. 3 // 2
6. -3 // 2
7. 11 % 2
8. 2 ** 10
9. 8 ** (1/3)

Wynik programu: Operacje zwracały wyniki zgodne z zasadami matematyki. Operator dzielenia (/) zawsze zwracał wynik
typu float, nawet przy dzieleniu liczb całkowitych. Operator // zwracał część całkowitą z dzielenia, a operator %
zwracał resztę. Potęgowanie przy użyciu ** umożliwiało obliczanie zarówno potęg całkowitych, jak i ułamkowych.

Analiza wyników: Python automatycznie dopasowuje typ danych do rodzaju działania. Działania z użyciem liczb rzeczywistych
zwracają wynik float, natomiast działania czysto całkowite (np. 1 + 2) zwracają wynik typu

int. Operator // wykonuje
dzielenie całkowite, a wynik jest zaokrąglany w dół (floor division), co widać w przypadku -3 // 2, które zwraca -2.

Wnioski: Python jest językiem dynamicznie typowanym, co oznacza, że typ danych nie musi być deklarowany jawnie.
Zrozumienie działania operatorów arytmetycznych jest kluczowe do dalszego programowania i tworzenia algorytmów numerycznych.

Zadanie 2

Cel zadania: Celem zadania było poznanie funkcji print(), służącej do wyświetlania tekstu i wyników obliczeń w konsoli.
Ćwiczenie miało na celu zrozumienie podstawowej komunikacji programu z użytkownikiem oraz utrwalenie pracy ze zmiennymi typu string.

Przebieg zadania: Utworzono zmienną tekstową o nazwie 'uczelnia', do której przypisano ciąg znaków. Następnie przy użyciu funkcji
print() wypisano zawartość tej zmiennej w konsoli. Sprawdzono różne warianty użycia funkcji print(), w tym możliwość łączenia
tekstów oraz użycia znaku nowej linii.

Kod programu:
```
uczelnia = "Studiuję na WSIiZ"
print(uczelnia)
```

Wynik programu: Konsola wyświetliła komunikat: Studiuję na WSIiZ.

Analiza wyników: Funkcja print() wypisuje wartości przekazane jako argumenty w postaci tekstowej. Może przyjmować wiele
argumentów rozdzielonych przecinkami, a także modyfikować zakończenie linii poprzez parametr end.

Wnioski: Funkcja print() jest jednym z najczęściej używanych narzędzi w Pythonie. Umożliwia szybkie testowanie kodu oraz
czytelne komunikowanie wyników użytkownikowi.

Zadanie 3

Cel zadania: Celem było poznanie sposobu przechowywania danych w zmiennych oraz ich używania w złożonych komunikatach
tekstowych. Ćwiczenie miało pokazać znaczenie interpolacji tekstu i formatu f-stringów w

Pythonie.

Przebieg zadania: Utworzono trzy zmienne – imię, wiek i wzrost. Następnie wykorzystano f-stringi, aby zbudować czytelny komunikat
z wykorzystaniem wartości przechowywanych w zmiennych. Użyto funkcji print() do wypisania danych w sformatowany sposób.

Kod programu:
```
imie = 'Jan'
wiek = 20
wzrost = 178
print(f"Nazywam się {imie} i mam {wiek} lat.")
print(f"Mój wzrost to {wzrost} cm.")
```

Wynik programu:
Nazywam się Jan i mam 20 lat.
Mój wzrost to 178 cm.

Analiza wyników: f-stringi w Pythonie umożliwiają wstawianie wartości zmiennych bez konieczności konwersji typów.
Dzięki temu kod jest bardziej czytelny i mniej podatny na błędy. Ułatwia to budowę dynamicznych komunikatów tekstowych.

Wnioski: F-stringi stanowią nowoczesny i efektywny sposób formatowania tekstu, pozwalający na szybkie tworzenie komunikatów
z danymi dynamicznymi.

Zadanie 4

Cel zadania: Celem zadania było praktyczne zastosowanie podstawowych operacji arytmetycznych w celu obliczenia ceny po rabacie.
Dodatkowym celem było nauczenie się kontrolowania precyzji wyświetlania wyników liczbowych.

Przebieg zadania: Zdefiniowano zmienne: cena (39.99) i rabat (0.2). Następnie obliczono cenę po uwzględnieniu rabatu,
korzystając ze wzoru: cena_po_rabacie = cena * (1 - rabat). Wynik sformatowano do dwóch miejsc po przecinku.

Kod programu:
```
cena = 39.99
rabat = 0.2
cena_po_rabacie = cena * (1 - rabat)
```

print(f"Cena po rabacie: {cena_po_rabacie:.2f} PLN")

Wynik programu: Cena po rabacie: 31.99 PLN.

Analiza wyników: Obliczenia wykazały, że rabat został poprawnie uwzględniony. Python pozwala na kontrolę liczby miejsc
po przecinku poprzez formatowanie ciągów znaków.

Wnioski: Python umożliwia wykonywanie obliczeń finansowych z zachowaniem dokładności oraz formatowania wyników.

Zadanie 5

Cel zadania: Celem zadania było stworzenie prostego programu wykorzystującego dane wejściowe użytkownika oraz podstawowe działania matematyczne do obliczeń geometrycznych.

Przebieg zadania: Program pobierał długości boków prostokąta od użytkownika, a następnie obliczał jego pole i obwód
zgodnie ze wzorami: pole = a * b oraz obwód = 2 * (a + b). Dane wejściowe konwertowano na typ float, aby umożliwić
obliczenia z wartościami rzeczywistymi.

Kod programu:

```
a = float(input("Podaj długość boku a: "))
b = float(input("Podaj długość boku b: "))
pole = a * b
obwod = 2 * (a + b)
print(f"Pole: {pole}, Obwód: {obwod}")
```

Wynik programu: Dla danych wejściowych a=5, b=3 wyniki były następujące: Pole: 15, Obwód: 16.

Analiza wyników: Program poprawnie wykorzystał funkcję input() do interakcji z użytkownikiem. Użytkownik mógł wprowadzić
wartości, które były dynamicznie przetwarzane. W praktyce jest to przykład prostego programu obliczeniowego.

Wnioski: Umiejętność przetwarzania danych wejściowych jest kluczowa w programowaniu interaktywnym.
Zadanie wprowadziło podstawy pracy z danymi użytkownika i arytmetyką w Pythonie.

Zadanie 6 – Obliczanie zużycia paliwa i kosztów podróży

Cel zadania: Celem było opracowanie programu obliczającego zużycie paliwa i koszt przejazdu samochodem na podstawie
danych podanych przez użytkownika. Zadanie miało charakter praktyczny – symulowało obliczenia stosowane w aplikacjach motoryzacyjnych.

Przebieg zadania: Program pobierał drogę w kilometrach oraz średnie spalanie w litrach na 100 km. Następnie obliczał zużycie paliwa
i koszt podróży przy stałej cenie 6.5 zł/litr. Zastosowano operacje matematyczne oraz formatowanie wyników za pomocą f-stringów.

Kod programu:

```
droga = float(input("Podaj drogę (km): "))
spalanie = float(input("Podaj spalanie (l/100km): "))
paliwo = (droga / 100) * spalanie
koszt = paliwo * 6.5
print(f"Zużycie paliwa: {paliwo} l, Koszt: {koszt} zł")
```

Wynik programu: Dla drogi 300 km i spalania 7 l/100 km wyniki były następujące: Zużycie paliwa: 21.0 l, Koszt: 136.5 zł.

Analiza wyników: Wyniki potwierdziły poprawność działania programu. Użytkownik mógł wprowadzać różne dane,
a program dynamicznie obliczał koszty.

Wnioski: Skrypt ten stanowi przykład praktycznego zastosowania Pythona do obliczeń ekonomicznych w życiu codziennym.

Zadanie 6a – Losowa droga i wprowadzenie ceny paliwa

Cel zadania: Celem było rozszerzenie programu o możliwość generowania losowej długości drogi oraz interaktywne wprowadzanie ceny paliwa.
Dzięki temu użytkownik mógł przeprowadzać symulacje w różnych warunkach.

Przebieg zadania: Do programu zaimportowano bibliotekę random i użyto funkcji randint() do losowania długości trasy
z określonego zakresu (50–500 km). Użytkownik wprowadzał spalanie oraz aktualną cenę paliwa. Program obliczał zużycie
i koszt przejazdu, a wyniki były formatowane z dokładnością do dwóch miejsc po przecinku.

Kod programu:

```
import random
droga = random.randint(100, 1000)
spalanie = float(input("Podaj spalanie (l/100km): "))
cena = float(input("Podaj cenę paliwa: "))
paliwo = (droga / 100) * spalanie
koszt = paliwo * cena
print(f"Droga: {droga} km, Zużycie: {paliwo:.2f} l, Koszt: {koszt:.2f} zł")
```

Wynik programu: Program generował różne długości trasy przy każdym uruchomieniu, np. 327 km lub 451 km.
Dzięki temu użytkownik mógł testować różne scenariusze kosztów podróży.

Analiza wyników: Użycie modułu random pozwoliło na dynamiczne testowanie programu. Funkcje pseudolosowe zwiększają elastyczność kodu i umożliwiają modelowanie zjawisk o charakterze zmiennym.

Wnioski: Program w tej formie jest bardziej realistyczny i praktyczny. Pokazuje możliwości Pythona w zakresie symulacji,
losowości oraz interakcji z użytkownikiem.