Budowanie Własnych Narzędzi

Agenda:

- **Krótkie przypomnienie:** Potrafimy modelować dane (listy, słowniki).
- **Nowy problem:** Nasz kod się powtarza (Zasada **DRY** - Don't Repeat Yourself).
- **Rozwiązanie: Funkcje** - nasze pierwsze reużywalne narzędzia.
- **Anatomia funkcji: def** i wywoływanie.
- **Ćwiczenie 1:** "Witaj, Funkcjo!" - definiujemy i wywołujemy pierwszą funkcję.
- **Komunikacja z funkcją**: Parametry i argumenty.
- **Ćwiczenie 2:** "Elastyczny Asystent" - funkcja, która przyjmuje dane.
- **Odbieranie wyników:** Instrukcja return.
- **Ćwiczenie 3:** "Kalkulator" - funkcja, która zwraca wynik.
- Zadania do samodzielnej pracy.

Ćwiczenie 1 - "Witaj, Funkcjo!"

Cel: Zdefiniowanie i wywołanie pierwszej, prostej funkcji.

Anatomia funkcji:

- **def nazwa_funkcji**(): - definicja (przepis).
- **nazwa_funkcji()** - wywołanie (użycie przepisu).

Zadanie:

- Zdefiniuj funkcję o nazwie **wyswietl_powitanie.**
- Wewnątrz funkcji umieść kilka instrukcji **print**, które wyświetlą dowolny, wieloliniowy nagłówek powitalny.
- **Poza** definicją funkcji, wywołaj ją dwukrotnie.

Ćwiczenie 2: ”Elastyczny Asystent”

Problem: Jak opakować powtarzalny kod i przekazać mu dane z zewnątrz?

Rozwiązanie: Funkcje z parametrami

- **Parametr:** Zmienna w **definicji** funkcji, która czeka na wartość.
- **Argument:** Konkretna wartość przekazywana podczas **wywołania** funkcji.

Cel: Zdefiniowanie i wywołanie funkcji, która przyjmuje dane.

```
# 'imie' to PARAMETR
def przywitaj_imiennie(imie):
    print(f"Cześć, {imie}!")

# "Anna" to ARGUMENT
przywitaj_imiennie("Anna")
```

Zadanie:

a. Zdefiniuj funkcję **przedstaw_sie**, która przyjmuje jeden parametr (np. **nazwa_uzytkownika**).

b. Wewnątrz funkcji, użyj **print** i f-stringa, aby wyświetlić komunikat, np. Cześć, jestem [nazwa_uzytkownika]. Miło mi Cię poznać..

c. **Poza** funkcją, użyj **input(),** aby zapytać użytkownika o jego imię i zapisać je w zmiennej.

d. Wywołaj funkcję **przedstaw_sie**, przekazując jej imię pobrane od użytkownika jako **argument**.

Omówienie Ćwiczenia 2: ”Elastyczny Asystent”

Cel: Analiza rozwiązania i utrwalenie przepływu danych do funkcji.

Kluczowa koncepcja: Wartość ze zmiennej **imie_od_usera** jest kopiowana do parametru **nazwa_uzytkownika** w momencie wywołania funkcji.

Ćwiczenie 3: ”Kalkulator” – Funkcja, która zwraca wynik

Problem: Nasze funkcje do tej pory tylko coś wyświetlały. Nie mogliśmy użyć ich wyniku w dalszych obliczeniach.

Rozwiązanie: Instrukcja return

- **return** natychmiast kończy działanie funkcji i "oddaje" wartość w miejsce, gdzie funkcja została wywołana.

```
# Funkcja, która OBLICZA, a nie wyświetla
def oblicz_pole(a, b):
    pole = a * b
    return pole # Zwróć obliczoną wartość

# Wynik funkcji jest przypisywany do zmiennej
pole_pokoju = oblicz_pole(5, 4)
koszt_malowania = pole_pokoju * 20 # Możemy użyć
wyniku!
print(f"Koszt malowania: {koszt_malowania} zł")
```

Cel: Zdefiniowanie i wywołanie funkcji, która zwraca obliczoną wartość.

Zadanie:

a. Zdefiniuj funkcję **dodaj**, która przyjmuje dwa parametry (np. **liczba1, liczba2**).

b. Wewnątrz funkcji, oblicz sumę tych dwóch liczb.

c. Użyj **return**, aby zwrócić obliczoną sumę.

d. Poza funkcją, wywołaj ją z dwoma dowolnymi liczbami.

e. Wynik działania funkcji zapisz w zmiennej **wynik_dodawania**.

f. Wyświetl zawartość tej zmiennej.

Omówienie Ćwiczenia 3: ”Kalkulator”

Cel: Analiza rozwiązania i utrwalenie koncepcji zwracania wartości.

Kluczowa koncepcja: Funkcja **dodaj** nie wie i nie dba o to, co stanie się z jej wynikiem. Jej jedynym zadaniem jest **obliczyć i zwrócić**.

Zaawansowane Argumenty: Elastyczność i Czytelność

Argumenty Nazwane (Keyword Arguments):

- **Problem:** Przy wielu parametrach, łatwo pomylić kolejność.
- **Rozwiązanie:** Jawne przypisanie wartości do parametru po nazwie.
- **Zaleta:** Kolejność **nie ma znaczenia**, a kod staje się bardziej czytelny.

```
def opisz_osobe(imie, wiek, miasto):
    print(f"{imie}, lat {wiek}, z miasta {miasto}.")

# Wywołanie z argumentami nazwanymi – kolejność
dowolna!
opisz_osobe(wiek=30, miasto="Kraków", imie="Anna")
```

Parametry Domyślne (Default Arguments):

- **Problem:** Jak uczynić parametr opcjonalnym?
- **Rozwiązanie:** Nadanie wartości domyślnej w definicji funkcji.
- **Ważne:** Parametry z wartością domyślną muszą być na końcu!

```
def przywitaj(imie, powitanie="Cześć"):
    print(f"{powitanie}, {imie}!")

przywitaj("Piotr") # Użyje domyślnego powitania
przywitaj("Anna", powitanie="Dzień dobry")
# Nadpisujemy domyślne
```

Ćwiczenie 4 – “Elastyczny Generator Raportów”

Cel: Samodzielne stworzenie elastycznej funkcji z użyciem argumentów nazwanych i domyślnych.

Problem: Stwórz funkcję, która generuje raport o użytkowniku, ale niektóre dane są opcjonalne.

Zadanie:

a. Zdefiniuj funkcję **generuj_raport**, która przyjmuje parametry:

- **imie** (obowiązkowy).
- **stanowisko** (opcjonalny, z wartością domyślną **"Pracownik").**
- **miasto** (opcjonalny, z wartością domyślną **"Nieznane").**

b. Wewnątrz funkcji, wyświetl sformatowany, wieloliniowy raport.

c. **Poza** funkcją, wywołaj ją **trzy razy**, aby przetestować jej elastyczność:

- Tylko z obowiązkowym argumentem **imie**.
- Z argumentem **imie** i podanym **miastem** (użyj argumentu nazwanego).
- Ze wszystkimi argumentami, podanymi w dowolnej kolejności (użyj argumentów nazwanych).

Wskazówka: Pamiętaj, że parametry z wartością domyślną muszą być na końcu listy parametrów w definicji funkcji.

Omówienie Ćwiczenia 4: "Elastyczny Generator Raportów"

Cel: Analiza rozwiązania i utrwalenie zastosowania argumentów domyślnych i nazwanych.

Zadanie 1: “Uniwersalny Kalkulator Pola”

Cel: Utrwalenie wiedzy o funkcjach, parametrach (w tym domyślnych) i zwracaniu wartości.

Problem: Stwórz jedną funkcję, która potrafi obliczyć pole różnych figur geometrycznych.

Kroki do wykonania:

a. Zdefiniuj funkcję **oblicz_pole**, która przyjmuje parametry:

- **figura**
  (obowiązkowy string, np. "prostokat", "kolo").
- **a, b, r**
  (opcjonalne, z wartościami domyślnymi **0**).

b. Wewnątrz funkcji, użyj **if/elif/else**, aby sprawdzić, jaką figurę podano.

c. Dla "prostokata" funkcja ma zwrócić **a * b**.

d. Dla "koła" funkcja ma zwrócić pole koła o promieniu **r** (wzór: $\pi * r^2$). Użyj **math.pi**.

e. Jeśli podano nieznaną figurę, funkcja ma zwrócić **None**.

f. **Poza** funkcją, przetestuj jej działanie dla obu figur i wyświetl wyniki.

Omówienie Zadania 1: "Uniwersalny Kalkulator Pola"

Cel: Analiza rozwiązania, które wykorzystuje parametry domyślne i logikę warunkową do stworzenia elastycznej funkcji.

Zadanie 2 - "Generator Raportów, który zwraca wynik"

Cel: Zastosowanie zasady jednej odpowiedzialności (SRP) poprzez oddzielenie logiki tworzenia danych od ich prezentacji.

Problem: Zmodyfikuj funkcję **generuj_raport** z poprzedniego ćwiczenia (Ćwiczenie 4).

Zadanie:

a. Otwórz swój kod z "Elastycznym Generatorem Raportów".

b. Zmień logikę funkcji tak, aby **nie używała print().**

c. Zamiast tego, funkcja ma **zbudować** jeden, wieloliniowy string zawierający cały raport.

d. Na końcu, funkcja ma zwrócić ten gotowy string za pomocą **return**.

e. **Poza** funkcją, wywołaj ją, 'złap' zwrócony raport do zmiennej i dopiero wtedy go wyświetl za pomocą **print().**

Wskazówka: Możesz budować wieloliniowy string, dodając do siebie kolejne linie z użyciem `+=` i znaków nowej linii **\n**.

Omówienie Zadania 2 (Refaktoryzacja)

Cel: Analiza rozwiązania, które oddziela logikę tworzenia danych od ich prezentacji.

Profesjonalne funkcje

Gdzie jesteśmy? Potrafimy budować proste funkcje.

Kolejny cel: Uzupełnić wiedzę z wykładu i nauczyć się pisać funkcje, które są elastyczne, bezpieczne i profesjonalnie udokumentowane.

Agenda (kontynuacja):

a. **Profesjonalna dokumentacja**: Jak pisać docstringi?
b. **Ćwiczenie 5:** Dokumentujemy nasz "Kalkulator Pola".
c. **Zwracanie wielu wartości:** Magia krotek i rozpakowywania.
d. **Ćwiczenie 6:** Funkcja, która zwraca imię i nazwisko.
e. **Argumenty dowolnej długości: *args i **kwargs.**
f. **Ćwiczenie 7:** Funkcja, która sumuje dowolną liczbę liczb.
g. **Najważniejsza pułapka:** Problem mutowalnych argumentów domyślnych.
h. Zadania do samodzielnej pracy.

Profesjonalna Dokumentacja: Docstringi

Problem: Jak opisać, co robi funkcja, co przyjmuje i co zwraca, aby inni (i my sami w przyszłości) mogli z niej łatwo korzystać?

Rozwiązanie: Docstring - specjalny komentarz umieszczony jako pierwsza linia w ciele funkcji.

Składnia: Tekst w potrójnych cudzysłowach """ ... """.

Rozszerzonie PEP 257 (Google Sheet Style):

- Pierwsza linia: Krótkie, jednowierszowe podsumowanie.
- Pusta linia.
- Sekcja **Args:** – Opis każdego argumentu
- Sekcja **Returns:** – Opis zwracanej wartości

Cel: Dodanie profesjonalnej dokumentacji do istniejącej funkcji.

Zadanie:

a. Otwórz swój kod z "Uniwersalnym Kalkulatorem Pola" (Zadanie 1 z poprzedniej części).

b. Dodaj do funkcji **oblicz_pole** kompletny **docstring** zgodny z konwencją PEP 257.

c. Opisz, co robi funkcja, jakie przyjmuje argumenty i co zwraca.

d. **Poza** funkcją, użyj wbudowanej funkcji **help()**, aby wyświetlić swoją dokumentację: **help(oblicz_pole).**

Omówienie Ćwiczenia 5: Profesjonalna Dokumentacja

Cel: Analiza poprawnego docstringu i jego praktycznego zastosowania.

Ćwiczenie 6: Zwracanie Wielu Wartości

Problem: Jak funkcja może zwrócić więcej niż jedną wartość? (np. imię i nazwisko, min i max)

Rozwiązanie: Funkcja może zwrócić jedną rzecz - **krotkę** zawierającą wiele wartości.

Cel: Zdefiniowanie i wywołanie funkcji, która zwraca wiele wartości.

Zadanie:

- Zdefiniuj funkcję **rozdziel_imie_nazwisko**, która przyjmuje jeden argument **imie_i_nazwisko** (np. "Jan Kowalski").
- Wewnątrz funkcji, użyj metody **.split()** do rozdzielenia tekstu.
- Użyj **return**, aby zwrócić jednocześnie imię i nazwisko.
- **Poza** funkcją, wywołaj ją i przypisz zwrócone wartości do dwóch osobnych zmiennych (**imie**, **nazwisko**).
- Wyświetl te dwie zmienne, aby potwierdzić, że rozdzielenie się udało.

```
def znajdz_min_max(liczby):
    # ... logika ...
    return min(liczby), max(liczby) # krotka

# Sposób 1: Odebranie krotki
wynik = znajdz_min_max([1, 5, -2])
print(wynik[0]) # -> -2

# Sposób 2 (Pythoniczny!): Rozpakowanie wyniku
min_val, max_val = znajdz_min_max([1, 5, -2])
print(f"Min: {min_val}, Max: {max_val}")
```

Omówienie Ćwiczenia 6 (Zwracanie Wielu Wartości)

Cel: Analiza rozwiązania i utrwalenie wzorca zwracania i rozpakowywania wielu wartości.

Kluczowa koncepcja: return imie, nazwisko tworzy krotkę, a **zmienna1, zmienna2 = funkcja()** elegancko ją rozpakowuje.

Ćwiczenie 7: Argumenty Pozycyjne Dowolnej Długości (*args)

Problem: Jak napisać funkcję, która przyjmie dowolną liczbę argumentów (np. funkcja print lub funkcja sumująca)?

Rozwiązanie: *args

- **Gwiazdka** * przed nazwą parametru zbiera wszystkie "nadmiarowe" argumenty **pozycyjne** do **krotki** (tuple).
- Nazwa **args** to tylko konwencja (od arguments).

Cel: Zdefiniowanie i wywołanie funkcji, która przyjmuje dowolną liczbę argumentów.

Składnia:

```
def pokaz_argumenty(pierwszy, *reszta):
     print(f"Pierwszy argument: {pierwszy}")
     print(f"Reszta zebrana do krotki: {reszta}")

pokaz_argumenty("A", "B", "C", "D")
```

Zadanie:

- Zdefiniuj funkcję **zsumuj_wszystko**, która przyjmuje dowolną liczbę argumentów liczbowych (np. ***liczby**).
- Wewnątrz funkcji, użyj wbudowanej funkcji **sum()** na krotce, aby obliczyć sumę wszystkich podanych liczb.
- Użyj **return**, aby zwrócić obliczoną sumę.
- **Poza** funkcją, przetestuj ją, wywołując z różną liczbą argumentów, np. zsumuj_wszystko(1, 2, 3) i zsumuj_wszystko(10, 20, 30, 40, 50).

Omówienie Ćwiczenia 7: “Sumowanie Wszystkiego”

Cel: Analiza rozwiązania i utrwalenie wzorca ***args** do zbierania argumentów pozycyjnych.

Kluczowa koncepcja: ***liczby** spakowało wszystkie argumenty pozycyjne do krotki **liczby**. Wewnątrz funkcji, **liczby** to zwykła krotka.

Ćwiczenie 8: Argumenty Nazwane Dowolnej Długości (**kwargs)

Problem: Jak napisać funkcję, która przyjmie dowolną liczbę argumentów **nazwanych**?

Rozwiązanie: **kwargs

- Dwie gwiazdki ** przed nazwą parametru zbierają wszystkie "nadmiarowe" argumenty **nazwane** do **słownika (dict).**
- **Nazwa kwargs to tylko konwencja**
  (od keyword arguments).

Cel: Zdefiniowanie i wywołanie funkcji, która przyjmuje dowolną liczbę argumentów nazwanych.

Składnia:

```
def pokaz_dane(**dane):
    print(f"Otrzymano słownik: {dane}")
    for klucz, wartosc in dane.items():
        print(f"- {klucz}: {wartosc}")

pokaz_dane(imie="Jan", wiek=40, miasto="Warszawa")
```

Zadanie:

- Zdefiniuj funkcję **generuj_raport**, która przyjmuje dowolną liczbę argumentów nazwanych (np. ****szczegoly**).
- Wewnątrz funkcji, użyj pętli **for** i metody **.items(),** aby wyświetlić każdą parę **klucz: wartość** z otrzymanego słownika.
- **Poza** funkcją, przetestuj ją, wywołując z różnymi zestawami argumentów nazwanych, np. *generuj_raport(status = "Aktywny", punkty = 150)*
  i
  generuj_raport(imie = "Anna", kraj = "Polska", wiek = 30).

Omówienie Ćwiczenia 8: Generator raportów

Cel: Analiza rozwiązania i utrwalenie wzorca ****kwargs** do zbierania argumentów nazwanych.

Kluczowa koncepcja: ****szczegoly** spakowało wszystkie argumenty nazwane do słownika **szczegoly**. Wewnątrz funkcji, **szczegoly** to zwykły słownik.

Najważniejsza Pułapka: Mutowalne Argumenty Domyślne

Problem: Wartość domyślna jest tworzona

TYLKO RAZ, podczas definicji funkcji.

Poprawny wzorzec (IDIOM):

```
def dodaj_element_poprawnie(element, lista=None):
    if lista is None:
        lista = []
    lista.append(element)
    print(f"Lista wewnątrz funkcji: {lista}")
```

Antywzorzec (NIE RÓB TEGO!):

```
def dodaj_element(element, lista=[]):
    lista.append(element)
    print(f"Lista wewnątrz funkcji: {lista}")

print("Pierwsze wywołanie:")
dodaj_element(1) # Oczekujemy: [1]

print("\nDrugie wywołanie:")
dodaj_element(2) # Oczekujemy: [2], a dostajemy...
[1, 2]!
```

Funkcje to też obiekty! (Obywatele pierwszej kategorii)

W Pythonie funkcje są traktowane jak każde inne dane (liczby, stringi, listy).

Co to oznacza w praktyce?

a. Można przypisać funkcję do zmiennej.

b. Można przekazać funkcję jako argument do innej funkcji.

Przykład:

```
def przywitaj(imie):
    return f"Cześć, {imie}!"

# Przypisanie funkcji do zmiennej, BEZ NAWIASÓW!
pozdrowienie = przywitaj
print(pozdrowienie("Anna")) # Wywołanie przez nową
nazwę
```

Wniosek: To fundament dla zaawansowanych technik, takich jak dekoratory, które poznamy w przyszłości.

Ćwiczenie 9: “Kalkulator Strategiczny”

Cel: Praktyczne zastosowanie funkcji jako argumentu (funkcje wyższego rzędu).

Problem: Stwórz uniwersalny kalkulator, którego działanie można zmieniać, podając mu różne funkcje operacji.

Zadanie:

a. Zdefiniuj dwie proste funkcje**: dodaj(a, b)** i **odejmij(a, b)**. Każda z nich ma przyjmować dwie liczby i **zwracać** wynik.

b. Zdefiniuj główną funkcję **wykonaj_operacje(a, b, funkcja_operacji).**

c. Wewnątrz **wykonaj_operacje**, wywołaj przekazaną **funkcja_operacji** na argumentach **a** i **b** i zwróć jej wynik.

d. **Poza** funkcjami, przetestuj swoje rozwiązanie:

- Wywołaj **wykonaj_operacje**, przekazując jej funkcję **dodaj**.
- Wywołaj **wykonaj_operacje**, przekazując jej funkcję **odejmij**.
- Wyświetl oba wyniki.

Omówienie Ćwiczenia 9: “Kalkulator Strategiczny”

Cel: Analiza rozwiązania i utrwalenie wzorca przekazywania funkcji jako argumentu.

Kluczowa koncepcja: Funkcja **wykonaj_operacje** nie wie, co robi. Wie tylko, jak użyć narzędzia (**funkcja_operacji**), które jej podano.

Podsumowanie

Co dzisiaj zrobiliśmy?

- Zrozumieliśmy zasadę **DRY** (Don't Repeat Yourself).
- Nauczyliśmy się tworzyć własne, reużywalne narzędzia (funkcje).
- Opanowaliśmy komunikację z funkcjami (**parametry, argumenty, return**).
- Stworzyliśmy elastyczne funkcje (**domyślne, *args, **kwargs**).
- Nauczyliśmy się profesjonalnie je dokumentować (**docstringi**).
- Zrozumieliśmy, że funkcje to obiekty, które można przekazywać.

Kluczowa lekcja: Dobrze napisana funkcja to fundament czystego, zorganizowanego i łatwego w utrzymaniu kodu.

Na następnych zajęciach:

- Zgłębimy **zaawansowane mechanizmy funkcji**: zasięg zmiennych, dekoratory i generatory.
- Zbudujemy nasz pierwszy, uniwersalny dekorator mierzący czas.