

PBL2 - Przykładowe Pytania Egzaminacyjne**C/μC/STM**

- 1. Jakie jest praktyczne znaczenie zastosowania atrybutu const w oprogramowaniu mikrokontrolerów?**

Odpowiedź:

1. **Optymalizacja pamięci Flash (ROM):** Dane oznaczone jako const są przechowywane w pamięci Flash, a nie w RAM, co **oszczędza cenną pamięć RAM**. Dane oznaczone jako const mogą być umieszczane w tej pamięci nieulotnej, co oszczędza cenną i często ograniczoną pamięć RAM. Jest to szczególnie przydatne dla stałych tabel, komunikatów tekstowych, stałych konfiguracyjnych i innych danych, które nie zmieniają się podczas działania programu.
 2. **Zwiększenie bezpieczeństwa i niezawodności:** Kompilator **uniemożliwia przypadkową zmianę** wartości const, co eliminuje błędy na etapie komplikacji i zwiększa integralność danych.
 3. **Lepsza czytelność:** Jasno komunikuje, że dana wartość jest stała, co **ułatwia zrozumienie kodu**.
 4. **Optymalizacja kodu:** Pozwala kompilatorowi na **dodatkowe optymalizacje**. Może na przykład wstawić wartość bezpośrednio do instrukcji maszynowej zamiast odwoływać się do pamięci, co przyspiesza wykonanie kodu i zmniejsza jego rozmiar.
 5. **Bezpieczeństwo wskaźników:** Atrybut const może być stosowany do wskaźników, aby określić, czy wskazywana wartość jest stała, czy sam wskaźnik jest stały. To pozwala na precyzyjną kontrolę nad dostępem do pamięci i zapobiega niezamierzonym modyfikacjom danych poprzez wskaźniki.
 6. **Inicjalizacja rejestrów peryferyjnych:** "Zwykle moduły peryferyjne inicjujemy tylko jeden raz – przy starcie programu". Wartości początkowe rejestrów sterujących są stałe i znane. Chociaż inicjalizacja odbywa się przez podstawienia, to same wartości użyte do inicjalizacji mogą być stałymi (const), co jest zgodne z zasadą inicjalizowania rejestrów tylko raz.
- 2. Opisać problem wynikający z modyfikacji stanu wyjść GPIO mikrokontrolera przy użyciu operacji logicznych oraz rozwiązanie eliminujące ten problem we współczesnych mikrokontrolerach.**

Odpowiedź:

Kiedy próbujesz zmienić stan pinu GPIO (np. włączyć diodę) za pomocą operacji typu rejestr $|=(1 \ll \text{pin})$, procesor wykonuje trzy kroki:

1. Odczytuje aktualny stan całego rejestru.
2. Modyfikuje ten stan, zmieniając tylko jeden bit.
3. Zapisuje zmodyfikowany stan z powrotem do rejestru.

Problem (zwany "hazardem R-M-W") polega na tym, że jeśli w trakcie tych trzech kroków (między odczytem a zapisem) coś innego (np. procedura obsługi przerwania lub inny rdzeń procesora) również zmieni ten sam rejestr, to Twoja zmiana może nadpisać i zniweczyć tę inną zmianę.

Rozwiązanie w nowych mikrokontrolerach:

Mają one specjalne, "bezpieczne" rejesty lub mechanizmy, które pozwalają na zmianę stanu pinu w jednym, nieprzerwanym kroku. Dzięki temu nie ma ryzyka kolizji i utraty danych. Przykłady to:

Rejestry "Set" i "Clear": Zapisanie wartości do rejestrów "Set" od razu ustawia bit, a do rejestrów "Clear" od razu go zeruje, bez wcześniejszego odczytu.

Aliases adresowe: Zapis do specjalnego adresu pamięci automatycznie wykonuje operację (np. ustawienie, zerowanie, przełączenie) na głównym rejestrze, również w jednym kroku.

3. Opisać format transmisji danych w szeregowym interfejsie asynchronicznym UART oraz wyjaśnić znaczenie bitów startu i stopu.

Odpowiedź:

W transmisji UART dane są wysyłane bit po bicie, bez osobnego zegara.

1. Stan spoczynku: Linia jest wysoka (logiczne 1).
2. Bit startu (0): Niski poziom sygnału informuje odbiornik o początku ramki danych i synchronizuje go.
3. Bity danych: Właściwe dane (najczęściej 8 bitów), przesyłane LSB najpierw.
4. Bit parzystości (opcjonalnie): Służy do wykrywania podstawowych błędów (np. czy liczba jedynek jest parzysta/nieparzysta).
5. Bity stopu (1 lub 2): Wysoki poziom sygnału, informuje o końcu ramki i przygotowuje linię na kolejną transmisję, zapewniając czas na przetwarzanie.

Znaczenie bitów startu i stopu:

Są kluczowe dla synchronizacji nadajnika i odbiornika w transmisji asynchronicznej. Bit startu sygnalizuje początek, a bity stopu koniec ramki, pozwalając odbiornikowi poprawnie odczytać dane i czekać na następną ramkę.

4. Jakie sekcje pamięci występują w oprogramowaniu mikrokontrolera.

Sekcje Pamięci Programu

Pamięć programu można podzielić na cztery główne sekcje:

- **Kod (Sekcja TEXT):** To obszar pamięci, w którym znajduje się skompilowany, wykonywalny **kod programu**, czyli instrukcje, które procesor ma wykonać. Jest to sekcja tylko do odczytu, co zapobiega przypadkowej modyfikacji kodu w trakcie działania programu. Na przykład, gdy piszesz funkcję main() w C, cały kod w jej wnętrzu, taki jak instrukcje printf("Witaj świecie!"), ląduje właśnie tutaj.
- **Dane Statyczne (Sekcja STATIC):** Ta sekcja przechowuje **globalne i statyczne zmienne**, które istnieją przez cały czas trwania programu. Dane te są inicjowane przed uruchomieniem programu. Dzieli się ją na dwie podsekcje: .data dla zmiennych zainicjowanych (np. int globalna_zmienna = 10;) i .bss dla zmiennych niezainicjowanych (np. static int licznik;, która zostanie domyślnie wyzerowana).
- **Dane Dynamiczne Automatyczne (Stos / Stack):** To obszar pamięci używany do przechowywania **lokalnych zmiennych funkcji** oraz **argumentów funkcji**. Pamięć na stosie jest alokowana i zwalniana automatycznie w kolejności LIFO (Last-In, First-Out) w miarę wywoływania i kończenia działania funkcji. Kiedy wywołujesz funkcję oblicz_sumę(int a, int b), zmienne a i b oraz wszelkie zmienne zadeklarowane wewnątrz oblicz_sumę (np. int wynik;) znajdują się właśnie na stosie.
- **Dane Dynamiczne Kontrolowane (Sterta / Heap):** Jest to obszar pamięci, który programista może **dynamicznie alokować i zwalniać** w trakcie działania programu. Służy do przechowywania danych, których rozmiar nie jest znany na etapie komplikacji lub które muszą istnieć dłużej niż pojedyncze wywołanie funkcji. Przykładowo, jeśli program wczytuje dane z pliku i nie wie z góry, ile ich będzie, może dynamicznie zaalokować pamięć na stercie za pomocą funkcji takich jak malloc() (w C) lub operatora new (w C++), a następnie musi jawnie zwolnić tę pamięć funkcją free() lub delete.

1. Kod (Sekcja TEXT)

Jest to statyczna sekcja pamięci, która istnieje przez cały czas życia programu i jest przeznaczona tylko do odczytu.

- **Zawartość:**

- Instrukcje programu.
- Stałe, takie jak tablice adresów dla instrukcji switch oraz literały.

2. Dane Statyczne (Sekcja STATIC)

Ta sekcja przechowuje dane, których czas życia jest równy czasowi życia całego programu. Ma stały rozmiar i może być podzielona na podsekcje:

- **.rodata (Stałe):** Przechowuje dane statyczne przeznaczone tylko do odczytu.
- **.data (Zmienne zainicjowane):** Zawiera zmienne, które mają nadane wartości początkowe i są dostępne zarówno do odczytu, jak i zapisu.
- **.bss (Zmienne niezainicjowane):** Przechowuje zmienne bez zdefiniowanych wartości początkowych. Nazwa pochodzi od "Block Started by Symbol". Sekcja ta jest zazwyczaj zerowana przed uruchomieniem programu.

3. Dane Dynamiczne Automatyczne (Stos / Stack)

Ta sekcja jest tworzona i usuwana w trakcie działania programu, co oznacza, że ma zmienny rozmiar.

- **Charakterystyka:**

- Tworzy **stos (stack)**.
- Służy do przechowywania argumentów i zmiennych lokalnych procedur.
- Kolejność usuwania danych jest zawsze odwrotna do kolejności ich tworzenia (LIFO).
- Każdy wątek w programie posiada swój własny, oddzielny stos.

4. Dane Dynamiczne Kontrolowane (Sterta / Heap)

Ta sekcja jest tworzona i usuwana jawnie przez programistę za pomocą funkcji takich jak malloc/free.

- **Charakterystyka:**

- Tworzy **sterzę (heap)**.
- Jej czas życia nie jest powiązany ze strukturą wywołań procedur.
- Kolejność tworzenia i usuwania danych jest dowolna.
- Jest to pamięć wspólnie dzielona przez wszystkie wątki danego zadania

5. Pamięć EEPROM serii 24xxx, jest połączona z mikrokontrolerem przy użyciu interfejsu I2C, działającego z szybkością transmisji 400 kb/s. Pamięć wymaga przesłania N-bajtowego adresu odczytywanej danej. Określić liczbę i znaczenie ramek i pakietów przesyłanych podczas transakcji odczytu z pamięci jednego oktetu danych. Oszacować czas potrzebny na zrealizowanie tej transakcji, przyjmując, że czas START i STOP jest taki sam, jak czas transmisji bitu danych.

2. Linux/Node-RED

2.1. Proszę omówić interfejsy (wejściowe/wyjściowe, przewodowe/bezprzewodowe) komputera jednopłytowego RaspberryPi 4. Proszę odnieść się do następujących zagadnień:

- a. Kategoria zastosowań – przykładowe urządzenia, które można podłączyć
- b. Przepustowość
- c. Odległości pomiędzy połączonymi urządzeniami

Odpowiedź:

1. Interfejsy USB (wyjście)

Raspberry Pi 4 posiada 2x USB 2.0 i 2x USB 3.0. To interfejsy **przewodowe**, zarówno wejścia, jak i wyjścia.

- a. **Zastosowania:** Klawiatury, myszy, pamięci USB (2.0); szybkie dyski zewnętrzne, kamery (3.0).
 - b. **Przepustowość:** USB 2.0: **480 Mbps**; USB 3.0: **5 Gbps**.
 - c. **Odległości:** Do **3-5 metrów** bez aktywnych przedłużaczy.
-

2. Interfejs Ethernet (wejście/wyjście)

Raspberry Pi 4 ma port **Gigabit Ethernet**. To interfejs **przewodowy**, zarówno wejście, jak i wyjście, do sieci.

- a. **Zastosowania:** Routery, przełączniki sieciowe, serwery NAS.
 - b. **Przepustowość:** **1 Gbps**.
 - c. **Odległości:** Do **100 metrów** (kable Cat5e/Cat6).
-

3. Interfejsy HDMI (wyjście)

Raspberry Pi 4 ma dwa porty **micro-HDMI**. To interfejsy **przewodowe**, wyjścia video i audio.

- a. **Zastosowania:** Monitory, telewizory, projektor (do 4K).
 - b. **Przepustowość:** Do **18 Gbps** (wystarczająca dla 4K). Gdy jeden monitor to 60Hz, a jak dwa to 30Hz.
 - c. **Odległości:** Do **10-15 metrów** bez wzmacniaczy.
-

4. Interfejsy GPIO (wejście/wyjście)

Raspberry Pi 4 ma 40-pinowe złącze **GPIO**. To interfejs **przewodowy**, zarówno wejście, jak i wyjście.

- a. **Zastosowania:** Czujniki (np. temperatury), diody LED, silniki, przyciski.
 - b. **Przepustowość:** Zależy od protokołu (np. I2C, SPI), brak jednej wartości w Gbps.
 - **I2C: 100 kbit/s** (Standard), **400 kbit/s** (Fast), do **1 Mbit/s** (Fast Mode Plus).
 - **SPI:** Od **kilku do dziesiątek Mbit/s** (maks. zegar do 125 MHz).
 - **UART:** Typowo **9600-115200 baud**, do **kilku Mbit/s**.
 - **Bit-Banging GPIO:** **Kilka do kilkudziesięciu kHz** (zależne od oprogramowania).
 - c. **Odległości:** Zazwyczaj do **kilkudziesięciu centymetrów**, dla **UARTa większe**.
-

5. Interfejs CSI (wejście)

Raspberry Pi 4 ma jeden port **CSI**. To interfejs **przewodowy**, wejście sygnału wideo.

- a. **Zastosowania:** Oficjalne moduły kamer Raspberry Pi.
 - b. **Przepustowość:** Wystarczająca dla wideo wysokiej rozdzielczości (np. 1080p).
 - c. **Odległości:** Krótkie taśmowe kable, zazwyczaj **15-30 cm**.
-

6. Interfejs DSI (wyjście)

Raspberry Pi 4 ma jeden port **DSI**. To interfejs **przewodowy**, wyjście sygnału wideo.

- a. **Zastosowania:** Oficjalne wyświetlacze dotykowe Raspberry Pi.
 - b. **Przepustowość:** Wystarczająca dla dedykowanych wyświetlaczy.
 - c. **Odległości:** Krótkie taśmowe kable, zazwyczaj **15-30 cm**.
-

7. Interfejs Wi-Fi(wejście/wyjście)

Raspberry Pi 4 ma wbudowany moduł **Wi-Fi 802.11ac**. To interfejs **bezprzewodowy**, zarówno wejście, jak i wyjście.

- a. **Zastosowania:** Routery bezprzewodowe, smartfony, tablety (dostęp do internetu, sieć lokalna).
 - b. **Przepustowość:** Do **150 Mbps** (na 2,4GHz) oraz **433 Mbps** (na 5 GHz).
 - c. **Odległości:** Kilkanaście/kilkadziesiąt metrów 2.4 GHz - większa: (do **50-70m w pomieszczeniach**); 5.0 GHz - mniejsza: (do **20-30m w pomieszczeniach**).
-

8. Interfejs Bluetooth (wejście/wyjście)

Raspberry Pi 4 ma wbudowany moduł **Bluetooth 5.0 BLE**. To interfejs **bezprzewodowy**, zarówno wejście, jak i wyjście.

- a. **Zastosowania:** Bezprzewodowe klawiatury/myszy, słuchawki, czujniki IoT.
 - b. **Przepustowość:** Do **2 Mbps**.
 - c. **Odległości:** Do kilkanaście metrów, MOŻE: **kilkudziesięciu metrów w pomieszczeniach** (nawet setek na otwartej przestrzeni).
-

9. Gniazdo Audio (3.5mm)

Raspberry Pi 4 ma jedno **gniazdo jack 3.5mm**. To interfejs **przewodowy**, zarówno wejście, jak i wyjście audio.

- a. **Zastosowania:** Słuchawki, głośniki, mikrofony.
 - b. **Przepustowość:** Wystarczająca dla wysokiej jakości dźwięku stereo.
 - c. **Odległości:** Do **kilku metrów** (np. 5-10m) bez utraty jakości.
-

10. Interfejs USB-C (Zasilanie) (wejście)

Port **USB-C** na Raspberry Pi 4 służy jako **przewodowe wejście zasilające**.

- **a. Zastosowania:** Podłączenie zasilaczy USB-C PD (Power Delivery), power banków.
- **b. Przepustowość:** Dotyczy mocy: wymaga **min. 15W (5V/3A)** dla stabilnej pracy.
- **c. Odległości:** Zazwyczaj do **1-2 metrów** dla stabilnego zasilania.

2.2.

2.1.1. Proszę omówić model bezpieczeństwa systemu Linux.

Odpowiedź:

1. Użytkownicy i grupy

- Każdy użytkownik ma unikalny UID i należy do grupy (GID).
- Kluczowe pliki:
 - /etc/passwd – dane użytkowników
 - /etc/shadow – zaszyfrowane hasła
 - /etc/group – informacje o grupach
- Administrator (root) ma pełne uprawnienia.

2. Prawa dostępu do plików

- Trzy poziomy: **właściciel, grupa, inni.**
- Trzy typy uprawnień:
 - r – odczyt
 - w – zapis
 - x – wykonanie (lub wejście do katalogu)

3. Zarządzanie uprawnieniami

- chmod – zmiana uprawnień
chmod 755 plik
- chown – zmiana właściciela
chown user:group plik
- chgrp – zmiana grupy

2.3.

- a. Zaznacz wszystkie pliki, które może zmodyfikować użytkownik bob należący do grupy gr2

```
drwxr-xr-x  1 pi    pi      11531 May 22  2018 dir1
drwx----- 1 joe   gr1     11531 May 22  2018 dir2
drwxr-x--- 1 pi    gr2     11531 May 22  2018 dir3
-rw-rw-rw-  1 joe   pi      11531 May 22  2018 file1
-rw-rw-r--  1 web   gr3     11531 May 22  2018 file2
-rw-r----- 1 joe   pi      11531 May 22  2018 file3
-rw------- 1 pi    gr2     11531 May 22  2018 file4
-rwxr-xr-x  1 pi    gr1     11531 May 22  2018 script1
-rwxr-x---  1 bob   gr1     11531 May 22  2018 script2
-rwx----- 1 pi    gr1     11531 May 22  2018 script3
-rw-rw-r--  1 joe   gr2     11531 May 22  2018 script4
```

Odpowiedź:

[d/-] [rwx] [rwx] [rwx]

			-- prawa dla innych (others)
		-----	prawa dla grupy (group)
	-----	-----	prawa dla właściciela (owner)
-----	-----	-----	typ pliku (d – directory (katalog) / „ – zwykły plik)

Symbol

Znaczenie

r Read (odczyt)

w Write (zapisywanie – dokonywanie zmian/usunięcie pliku)

x Execute (uruchomienie pliku/wejście do katalogu)

- Brak danego prawa

2.4.

2.4.1. Dane jest wyrażenie regularne **17[0123]* /tcp** Podkreśl tekst pasujący do podanego wyrażenia:

db-lsp	17500/tcp	# Dropbox LanSync Protocol
sgi-cad	17003/tcp	# Cluster Admin daemon
sgi-cmsd	17001/udp	# Cluster membership services daemon
sgi-crsd	17002/udp	
sgi-gcd	17003/udp	# SGI Group membership daemon
sgi-cad	17004/tcp	# Cluster Admin daemon

CZYLI: 17 musi być, potem dowolna ilość [0123] i potem /tcp musi być

2.4.2. napisz wyrażenie regularne pasujące do komentarzy znajdujących się na końcu wiersza w przykładzie z punktu a.

XD

2.5. Jakiego rodzaju interfejs użytkownika jest w edytorze systemu Node-RED

Odpowiedź:

Interfejs użytkownika w Node-RED to **graficzny, przeglądarkowy interfejs typu drag-and-drop**, pozwalający na intuicyjne tworzenie aplikacji IoT, automatyzacji i przetwarzania danych bez konieczności pisania kodu w tradycyjny sposób.

2.6. Procesy w systemie Linux. W odpowiedzi proszę odnieść się do następujących zagadnień: **Co to są procesy**

W systemie Linux **proces** jest instancją uruchomionego programu. Każdy proces ma swój własny, izolowany obszar pamięci, swoje zasoby (takie jak otwarte pliki, deskryptory sieciowe) oraz własny kontekst wykonania (stan procesora, rejestrów). System operacyjny zarządza procesami, przydzielając im czas procesora i inne zasoby, a także kontrolując ich cykl życia (tworzenie, wykonywanie, zakończenie).

Po ludzku: Proces w Linuxie to po prostu uruchomiony program, który ma swoje własne, wydzielone zasoby (pamięć, identyfikator) i działa niezależnie od innych, umożliwiając komputerowi wykonywanie wielu zadań jednocześnie.

2.6.1. Jakie są rodzaje procesów (przykłady)

- **Procesy pierwszoplanowe (foreground processes):** Są to procesy, które są bezpośrednio połączone z terminaliem użytkownika i wymagają jego interakcji. Kiedy uruchomisz program w terminalu, domyślnie działa on jako proces pierwszoplanowy. Terminal czeka na zakończenie tego procesu, zanim będzie można wprowadzić kolejne polecenie. **Przykład:** Uruchomienie edytora tekstu nano
- **Procesy działające w tle (background processes):** Są to procesy, które działają niezależnie od terminala, z którego zostały uruchomione. Nie blokują terminala, co pozwala użytkownikowi na kontynuowanie pracy w tym samym terminalu. Często są to usługi systemowe lub długotrwałe zadania. **Przykład:** Można uruchomić proces w tle, dodając znak & na końcu polecenia, np.
`moja_aplikacja &`

2.6.2. Strumienie standardowe związane z procesami

Każdy proces w systemie Linux, kiedy jest uruchamiany, ma domyślnie otwarte trzy standardowe strumienie I/O (wejścia/wyjścia):

- **Standardowe wejście (Standard Input - stdin):** Jest to strumień, z którego proces odczytuje dane wejściowe. Domyślnie jest to klawiatura terminala.
 - **Standardowe wyjście (Standard Output - stdout):** Jest to strumień, na który proces wypisuje swoje normalne dane wyjściowe. Domyślnie jest to ekran terminala.
 - **Standardowe wyjście błędów (Standard Error - stderr):** Jest to strumień, na który proces wypisuje komunikaty o błędach. Domyślnie również jest to ekran terminala.
- Strumienie te można przekierowywać, co pozwala na elastyczne sterowanie wejściem i wyjściem procesów (np. zapisywanie wyjścia do pliku zamiast wyświetlanego na ekranie).

2.6.3. Uprawnienia procesu (opcjonalnie)

- Każdy proces jest uruchamiany w kontekście konkretnego użytkownika i grupy.
- Proces dziedziczy uprawnienia od użytkownika, który go uruchomił. Oznacza to, że proces może wykonywać operacje tylko na tych plikach i zasobach, do których ma dostęp użytkownik-właściciel procesu.
- Istnieje specjalny użytkownik `root` (administrator), którego procesy mają pełne uprawnienia do systemu.

2.6.4. Podstawowe narzędzia systemowe do monitorowania/zarządzania procesami

`ps aux` – do wypisania procesów z dokładnymi danymi typu: kto uruchomił, ile zajmuje procesora, ile RAMu, status procesu (S – uśpiony, R – biegący, Z – zombie, Ss – systemowy), kiedy uruchomiony, czas CPU jaki zużył,

Typowe narzędzia to również `top`, `htop`, `kill`, `pkill` (głównie zabijanie procesów)

Python/μPython

1) Co wypisze program:

```
fruits = ["apple", "banana", "cherry", "orange", "kiwi",
"melon", "mango"]
print(fruits[:-1])
```

Odpowiedź:

['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon']

2) Do czego służą pliki `__init__.py` w pakietach pythona

Plik `__init__.py` w pakiecie Pythona służy do:

- Identyfikacji pakietu: Mówią Pythonowi, że dany katalog jest pakietem.
- Inicjalizacji pakietu: Wykonuje kod przy pierwszym zimportowaniu pakietu, np. ustawia zmienne, importuje podmoduły lub definiuje, co ma być dostępne przez `from pakiet import *`.
- Dzięki temu plikowi można importować moduły kodu (np. funkcje)

3) Do czego służą poniższe linie w kodzie języka Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

Odpowiedź:

`#!/usr/bin/env python3` mówi systemowi:

„Znajdź interpreter `python3` w aktualnym środowisku użytkownika (np. `venv`) i go użyj.

Określa, **jaki interpreter** ma być użyty do uruchomienia skryptu. Dzięki temu można uruchomić skrypt jak program, np. w terminalu: `./skrypt.py` zamiast `python3 skrypt.py`.

`#-*- coding: utf-8 -*-` oznacza, że plik źródłowy używa kodowania UTF-8 (dla poprawnej obsługi znaków).

4) Zdefiniuj konstruktor klasy Person przyjmujący i inicjujący pole name.

Odpowiedź:

```
class Person:
    def __init__(self, name):
        self.name = name
```

- `__init__` – to konstruktor, czyli metoda wywoływana automatycznie przy tworzeniu obiektu klasy.
- `self` – odnosi się do bieżącego obiektu (instancji).
- `self.name = name` – przypisuje wartość argumentu `name` do pola `name` obiektu.

5) Do czego służy w programach Python:

```
if __name__ == '__main__':
```

Odpowiedź:

Służy do określenia punktu wejścia programu w Pythonie, czyli do tego, czy dany plik jest uruchamiany bezpośrednio, czy importowany jako moduł.

Ten mechanizm pozwala **oddzielić logikę (funkcje) od interakcji z użytkownikiem**, dzięki czemu kod może być:

- testowany,
- wielokrotnie używany,
- czytelniejszy.

Do czego służy if __name__ == '__main__': w Pythonie?

Ten fragment kodu to **specjalny warunek**, który mówi Pythonowi:

1. "Czy to ja jestem głównym programem?" 🚀
 - Gdy uruchamiasz plik Pythona **bezpośrednio** (np. klikając w niego albo wpisując python moj_plik.py w terminalu), Python wewnętrznie nadaje mu specjalne "imię" __main__. Wtedy warunek if __name__ == '__main__': jest **prawdziwy**.
 - Jeśli natomiast ten sam plik jest tylko **importowany** do innego programu (żeby użyć jego funkcji), to Python nadaje mu inne "imię" (np. nazwę pliku, czyli moj_plik). Wtedy warunek jest **falszywy**.
2. "Wykonaj ten kod tylko, jeśli jestem główny!" ✨
 - Kod, który znajdzie się **wewnątrz** bloku if __name__ == '__main__':, wykona się **tylko wtedy**, gdy Twój plik jest uruchamiany jako główny program.
 - Dzięki temu możesz mieć w jednym pliku zarówno funkcje, które inni mogą importować i używać, jak i kod, który służy do uruchomienia tego pliku jako samodzielnego programu (np. do testowania albo do uruchomienia głównej logiki).

6) Co trzeba zrobić, aby skrypt z kodem w μPython uruchomił się automatycznie po włączeniu zasilania układu Raspberry Pi Pico

Odpowiedź:

Nazwać plik z kodem jako main.py

MicroPython automatycznie **szuka i uruchamia plik main.py** po starcie urządzenia.

7) Na czym polegają różnice między Pythonem a μPython-em?

Cechy	Python	MicroPython
System docelowy	Komputer/serwer	Mikrokontroler
Rozmiar	Duży	Lekki i zoptymalizowany
Zasoby sprzętowe	Wysokie wymagania	Minimalne wymagania
Funkcje języka	Pełny Python 3	Podzbior Pythona 3
Biblioteki	Bogate	Ograniczone, zorientowane na sprzęt
Przykładowe użycie	Aplikacje web, automatyzacja	Sterowanie diodą, odczyt czujnika, IoT

8) Co trzeba zrobić, żeby w naszym programie w języku Python korzystać z biblioteki `Balbinka` ?

Odpowiedź:

Zakładamy, że biblioteka jest dostępna w systemie zarządzania pakietami pip.

W terminalu wpisz: pip install balbinka

W kodzie Pythona dodaj na początku: import balbinka
lub jeśli chcesz zaimportować tylko konkretną klasę/funkcję: from balbinka import jakas_funkcja

Np. jeśli Balbinka ma funkcję powiedz(), możesz ją wywołać: balbinka.powiedz("Cześć!")

Jeśli Balbinka to Twoja własna biblioteka lokalna:

1. Umieść plik balbinka.py w tym samym katalogu co Twój skrypt.
2. W kodzie użyj zwykłego import balbinka.

Jeśli to folder (pakiet), musi on zawierać plik __init__.py.