CZĘŚĆ A – SZUKAMY I NAPRAWIAMY BŁĘDY

Zadanie A1 – Co się psuje przy złym wyborze?

1. Uruchom program.
2. Na ekranie powitalnym zamiast cyfry wpisz np. `x` albo `enter` bez cyfry.
3. Przepisz (lub zrób zrzut) komunikat błędu z konsoli.

Pytania:

a) W której funkcji pojawia się błąd?
b) Dlaczego użycie `int(input(...))` jest tutaj niebezpieczne?

Zadanie do kodu:
Popraw funkcję `ekranPowitalny()` tak, aby:

- nie wywalała programu, jeśli użytkownik wpisze coś innego niż liczba,
- w takiej sytuacji pokazywała komunikat typu:
  `warn("Podaj numer opcji (1–4).")`
- dopiero gdy użytkownik poda poprawny numer, funkcja zwraca tę wartość.

Zadanie A2 – Pusta lista runnerów

1. Uruchom program i od razu wybierz opcję „lista runnerów” (3), **nie tworząc wcześniej żadnego konta**.
2. Zobacz, co pojawia się na ekranie.

Pytania:

a) Czy program jasno komunikuje, że nie ma żadnych runnerów?
b) Jak może się poczuć użytkownik, który „nie widzi” żadnych danych?

Zadanie do kodu:
W funkcji `pokazRunnerow(runnerzy)`:

- dodaj sprawdzenie, czy lista `runnerzy` jest pusta,
- jeśli tak, wyświetl komunikat, np.:
  `print(warn("Brak zarejestrowanych runnerów w systemie."))`
- dopiero gdy lista nie jest pusta, wypisuj ich dane.

Zadanie A3 – Zakładanie nowego konta

Spójrz na funkcję `nowaOsoba(klienci)`.

1. Co się dzieje, jeśli wpiszesz istniejący już login?
2. Czy program pozwala od razu spróbować ponownie, czy trzeba wracać do menu?

Zadanie do kodu:
Przerób `nowaOsoba(klienci)` tak, aby:

- działała w pętli dopóki użytkownik nie poda **poprawnych danych**
  (login nie zajęty, hasło i login dłuższe niż 1 znak),

- dawała możliwość **przerwania tworzenia konta**, np. po wpisaniu specjalnego loginu `q`/`quit`.

Przykład (zapis ogólny, nie musisz używać dokładnie tych słów):

Jeśli użytkownik wpisze `q` jako login – funkcja kończy się bez dodawania osoby.

Zadanie A4 – Misje, które nic nie robią

W `menu_runner(osoba)` widzisz listę misji:

```
print("1) Szept Sieci")
print("2) Karta Widmo")
print("3) Węzeł Nimbus")
print("4) Odbicie Zastawu")
print("5) Szlak Nanopaczek")
```

Obecnie po wyborze `1` wywoływane jest:

```
if m == "1":
    szeptSieci()
pauza("ENTER, by wrócić...")
```

Reszta opcji nic nie robi.

Zadanie do kodu:

1. Dodaj obsługę pozostałych opcji `2`–`5` w taki sposób, że:
   - na razie tylko wyświetlają tekst typu:
     `print(info("Ta misja jest w budowie..."))`
2. Dodaj obsługę sytuacji, gdy użytkownik wpisze coś spoza 1–5
   (np. komunikat o nieznanej misji).

CZĘŚĆ B – ROZWÓJ PROJEKTU (FABUŁA + MECHANIKA)

Teraz, gdy program działa stabilniej, czas rozwinąć samą grę.

Zadanie B1 – Ulepsz misję „Szept Sieci”

Aktualnie funkcja:

```
def szeptSieci():
    print("Szept sieci")
```

nic ciekawego nie robi.

Zadanie do kodu (kilka kroków):

1. Zmień nagłówek funkcji na:
2. `def szeptSieci(osoba):`

   tak, żeby misja miała dostęp do statystyk gracza.

3. W `menu_runner(osoba)` zmień wywołanie:
4. `szeptSieci()`

   na:
   `szeptSieci(osoba)`

5. W samej misji:
    - dodaj krótki opis sytuacji (kilka linijek tekstu),
    - daj graczowi **co najmniej 2 wybory** (np. „atakujesz serwer” vs „podsłuchujesz cicho”),
    - w zależności od wyboru **zmień pola** w słowniku `osoba`, np.:
        - `kredyty` +/–
        - `reputacja` +/–
        - `cieplo` (heat) +/–
6. Po zakończeniu misji wyświetl podsumowanie, np.:
   `print(info("Misja zakończona. Zaktualizowano Twoje statystyki."))`
   i użyj `pauza()`, żeby zatrzymać ekran.

Zadanie B2 – Własna misja: „Karta Widmo”

Stwórz nową funkcję:

```
def kartaWidmo(osoba):
    # opis fabularny
    # wybory gracza
    # zmiany w kredytach / reputacji / cieple
```

Wymagania:

- minimum **2 różne ścieżki** (decyzje) dla gracza,
- każda ścieżka w inny sposób zmienia statystyki gracza,
- przynajmniej w jednej opcji gracz **coś zyskuje**, a w innej **coś traci** (np. więcej kredytów, ale dużo „ciepła”).

Podłącz misję do menu:

- w `menu_runner(osoba)` dodaj:
- `elif m == "2":`
- `    kartaWidmo(osoba)`

Zadanie B3 – System „HEAT” (poziom zagrożenia)

W słowniku gracza znajduje się pole `cieplo` (heat).
Użyj go jako **paska ryzyka**.

Zadanie do kodu:

1. Po każdej misji (np. po wyjściu z `szeptSieci(osoba)` oraz `kartaWidmo(osoba)`) sprawdzaj wartość `osoba['cieplo']`.
2. Jeśli `cieplo` przekroczy ustalony poziom (np. `5` lub `10`):
    - wyświetl komunikat, że system bezpieczeństwa namierzył działania gracza,
    - zakończ sesję gracza (np. `break` w `menu_runner`, a nawet koniec programu).

Dodatkowe pytania (do odpowiedzi na kartce / w pliku):

- Jaki poziom HEAT wybrałeś jako „krytyczny”? Dlaczego?
- Czy są misje, które bardziej podnoszą HEAT niż inne?