KLUCZ ODPOWIEDZI · DLA NAUCZYCIELA

Sprawdzian z programowania

Grupa A

1. (1 pkt) **A**

2. (1 pkt) **A**

3. (1 pkt) **C**

4. (1 pkt) **A, B**

5. (1 pkt) **A, B, E**

6. (4 pkt) **1–A, 2–D, 3–C, 4–B** — po 1 pkt za parę

7. (2 pkt) Przykładowa odpowiedź: Słownik (dict) w Pythonie to struktura danych przechowująca pary klucz-wartość, np. `{'imie': 'Anna', 'wiek': 17}`. W przeciwieństwie do listy, w której elementy adresuje się kolejnym numerem indeksu (np. `lista[0]`), elementy słownika adresuje się poprzez unikalny klucz (np. `slownik['imie']`, co zwróci `'Anna'`), a nie przez pozycję w strukturze. Klucze słownika muszą być niemutowalne (np. napisy, liczby, krotki), a kolejność elementów w słowniku (od Python 3.7) odpowiada kolejności ich dodania, ale to klucz, nie kolejność, służy do wyszukiwania wartości.
Punktacja: Poprawne wyjaśnienie, że słownik przechowuje pary klucz-wartość, a dostęp do elementu odbywa się przez klucz, a nie przez indeks liczbowy (1 pkt) + poprawny przykład kodu (1 pkt). 2 pkt.

8. (2 pkt) Przykładowa odpowiedź: Zakres (scope) zmiennej to obszar kodu, w którym dana zmienna jest widoczna i można się do niej odwołać. Zmienna lokalna jest tworzona wewnątrz funkcji i istnieje tylko podczas jej wykonywania — po zakończeniu funkcji przestaje istnieć i nie jest widoczna na zewnątrz. Zmienna globalna jest zdefiniowana poza funkcjami, na poziomie modułu, i jest dostępna do odczytu wewnątrz każdej funkcji (modyfikacja wymaga słowa kluczowego `global`).
Punktacja: Poprawna definicja zakresu zmiennej (obszar programu, w którym zmienna jest widoczna i dostępna) — 1pkt. Poprawne wyjaśnienie różnicy: zmienna lokalna istnieje tylko wewnątrz funkcji, w której została utworzona, zmienna globalna jest dostępna w całym module — 1pkt. Maks. 2 pkt.

9. (3 pkt) Przykładowa odpowiedź: Pętla for przechodzi po elementach sekwencji lub zakresu, więc liczba obrotów jest znana z góry. Pętla while działa, dopóki warunek jest prawdziwy, i liczba obrotów zależy od danych. While jest naturalna np. przy wczytywaniu liczb od użytkownika aż do podania zera: nie wiadomo, ile liczb zostanie wpisanych, więc nie da się użyć gotowego zakresu.
Punktacja: Różnica: for iteruje po znanej sekwencji lub zakresie, while powtarza do spełnienia/niespełnienia warunku (1 pkt). Poprawny przykład użycia while, w którym liczba powtórzeń nie jest z góry znana (1 pkt). Uzasadnienie wyboru (1 pkt). 3 pkt.

10. (3 pkt) Przykładowa odpowiedź: Enkapsulacja: stan obiektu ukryty za interfejsem (pola prywatne, metody publiczne). Chroni integralność danych — użytkownik klasy używa metod, nie manipuluje bezpośrednio polami. Dziedziczenie: klasa potomna (np. Pies) dziedziczy pola i metody klasy bazowej (Zwierzę). Umożliwia wielokrotne użycie kodu i hierarchie typów. Polimorfizm: różne klasy mogą definiować własną wersję tej samej metody (np. wydaj_dzwiek()). Kod klienta wywołuje metodę na referencji do klasy bazowej — konkretna implementacja wybierana jest w czasie wykonania.
Punktacja: 1. Enkapsulacja: ukrywanie stanu wewnętrznego obiektu za interfejsem (gettery/settery, modyfikatory dostępu) (1 pkt). 2. Dziedziczenie: klasa pochodna przejmuje pola i metody klasy bazowej, może rozszerzać lub nadpisywać (1 pkt). 3. Polimorfizm: różne klasy mogą implementować tę samą metodę inaczej — kod klienta nie zna konkretnego typu (1 pkt).

Grupa B

1. (1 pkt) **C, D**

2. (1 pkt) **C**

3. (1 pkt) **C**

4. (1 pkt) **B**

5. (1 pkt) **A**

6. (4 pkt) **1–D, 2–A, 3–B, 4–C** — po 1 pkt za parę

7. (2 pkt) Przykładowa odpowiedź: Typ mutowalny to taki, którego zawartość można zmieniać po utworzeniu obiektu, bez tworzenia nowego obiektu w pamięci — przykładem jest `list`, do której można dodawać lub usuwać elementy metodą `append`/`remove`. Typ niemutowalny (np. `str` lub `tuple`) nie pozwala na zmianę swojej zawartości — każda operacja, która wygląda jak „modyfikacja" (np. konkatenacja stringów), w rzeczywistości tworzy zupełnie nowy obiekt w pamięci, a oryginalny pozostaje niezmieniony.
Punktacja: Poprawne wyjaśnienie różnicy: typ mutowalny pozwala zmienić zawartość obiektu bez tworzenia nowego obiektu w pamięci, a typ niemutowalny raz utworzony nie może zostać zmieniony — każda „modyfikacja" tworzy nowy obiekt (1 pkt). Podanie poprawnych przykładów, np. mutowalny: `list`, `dict`; niemutowalny: `str`, `tuple`, `int` (1 pkt). Łącznie 2 pkt.

8. (2 pkt) Przykładowa odpowiedź: Funkcja rekurencyjna to funkcja, która w swoim ciele wywołuje samą siebie, rozwiązując problem poprzez sprowadzenie go do mniejszej wersji tego samego problemu. Aby uniknąć nieskończonej rekurencji, funkcja musi zawierać: warunek bazowy (stopu) — sytuację, dla której funkcja zwraca wynik bezpośrednio, bez kolejnego wywołania samej siebie, oraz krok rekurencyjny, który z każdym wywołaniem przybliża problem do warunku bazowego (np. zmniejsza rozmiar danych wejściowych).
Punktacja: Poprawne wyjaśnienie pojęcia funkcji rekurencyjnej jako funkcji wywołującej samą siebie (1 pkt). Wskazanie dwóch koniecznych elementów: warunku bazowego (stopu) kończącego rekurencję oraz kroku rekurencyjnego, który przybliża problem do warunku bazowego (1 pkt). Łącznie 2 pkt.

9. (3 pkt) Przykładowa odpowiedź: Przekazywanie przez wartość oznacza, że funkcja otrzymuje kopię danych i zmiany wprowadzone wewnątrz funkcji nie wpływają na oryginał. Przekazywanie przez referencję oznacza, że funkcja otrzymuje odwołanie do tego samego obiektu w pamięci. W Pythonie dla typów niemutowalnych (np. int, string) każda „zmiana" wewnątrz funkcji tworzy nowy obiekt, więc zmienna w kodzie wywołującym pozostaje niezmieniona. Dla typów mutowalnych (np. listy, słowniki) funkcja otrzymuje odwołanie do tego samego obiektu, więc modyfikacja jego zawartości (np. `lista.append(x)`) jest widoczna również poza funkcją.
Punktacja: Poprawne ogólne wyjaśnienie różnicy — przez wartość kopiuje dane, przez referencję przekazuje odwołanie do tego samego obiektu (1 pkt) + poprawne wskazanie zachowania dla typów niemutowalnych w Pythonie — zmiana wewnątrz funkcji nie wpływa na zmienną zewnętrzną (1 pkt) + poprawne wskazanie zachowania dla typów mutowalnych — modyfikacja obiektu (np. listy) wewnątrz funkcji jest widoczna na zewnątrz (1 pkt). Maksymalnie 3 pkt.

10. (3 pkt) Przykładowa odpowiedź: Rozwiązanie iteracyjne wykorzystuje pętlę (np. for lub while) do wielokrotnego powtarzania obliczeń, natomiast rozwiązanie rekurencyjne polega na tym, że funkcja wywołuje samą siebie z coraz mniejszym podproblemem, aż osiągnie przypadek bazowy. Zaletą rekurencji jest często prostszy i bardziej czytelny zapis dla problemów, które mają naturalnie rekurencyjną strukturę (np. przechodzenie drzew). Wadą jest większe zużycie pamięci — każde wywołanie zajmuje miejsce na stosie wywołań, co przy dużej głębokości rekurencji może prowadzić do przepełnienia stosu.
Punktacja: Poprawne wyjaśnienie różnicy — iteracja używa pętli, rekurencja polega na wywoływaniu funkcji przez samą siebie (1 pkt) + trafna zaleta rekurencji, np. czytelniejszy zapis dla problemów o naturalnie rekurencyjnej strukturze (1 pkt) + trafna wada, np. większe zużycie pamięci (stos wywołań) i ryzyko przepełnienia stosu (1 pkt). Razem 3 pkt.

Skala ocen (orientacyjna, 19 pkt w każdej grupie)

bardzo dobry (5)	18–19 pkt
dobry (4)	15–17 pkt
dostateczny (3)	10–14 pkt
dopuszczający (2)	6–9 pkt
niedostateczny (1)	0–5 pkt

Progi: 90%, 75%, 50%, 30%. Progi w Twojej szkole mogą być inne. W zadaniach otwartych uznaj każdą odpowiedź poprawną merytorycznie, także sformułowaną inaczej niż przykład.

Ten sprawdzian online, z wyjaśnieniem każdej odpowiedzi: **matury-online.pl/sprawdzian/informatyka/programowanie**. Nauczyciele i korepetytorzy ułożą własną kartę z banku ponad 20 tys. zadań: **matury-online.pl/dla-korepetytorow**.