Zadanie 1

Pytanie: Wymień i krótko scharakteryzuj platformy , na których można uruchomić aplikację napisane w języku Swift

Platformy na których możemy uruchomić aplikacje napisane w języku Swift to oczywiście:

IOS - (system operacyjny przeznaczony dla iPhone'a oraz urządzeń docelowych firmy Apple np. IPod Docelowo najpopularniejsza platforma na której można uruchomić aplikacje napisane w języku Swift. Programiści mogą korzystać z frameworków takich jak UIKit (poznany na zajęciach ćwiczeniowych) albo SwifUI. Aplikacje odpalane na tej platformie mogą mieć charakter prostych narzędzi aż po zaawansowane aplikacje biznesowe czy nawet gry.

MacOS - (system operacyjny przeznaczony dla komputerów Maca). Tutaj programiści mogą tworzyć aplikacje z użyciem również SwiftUI również o dużym stopniu zaawansowania. Mogą to być gry aplikacje biznesowe , proste narzędzia , treści multimedialne itd.

Linux – system ten nie jest oficjalną platformą Apple ale w ramach ciekawostki można wspomnieć , że Swift będąc open-sourcem wykorzystywany jest na Linuxie do tworzenia aplikacji serwerowych , skryptów bądź narzędzi konsolowych.

W ramach usług oferowanych przez firmę Apple możemy dodać jeszcze , że platformy na których możemy uruchomić aplikację w języku Swift to TvOS (multimedia i rozrywka powiązana platforma to Apple TV czyli streaming) , WatchOS (system operacyjny powiązany z zegarkami/smartwatchami od Appla)

Zadanie 2

Pytanie: Porównaj ze sobą SwiftUI oraz UIKit w zakresie tworzenia interfejsu graficznego

Zacznijmy od podejście programistycznego – SwiftUI ma tak zwany deklaratywny charakter czyli zamiast mówić jak coś ma być wykonane mówimy czego oczekujemy od frameworka a on zajmuje się resztą. Deklarujemy w nim widoki (małe niezależne interfejsy użytkownika takie jak przyciski , tekst , obrazki) możemy dodawać do nich modyfikatory itd.

UIKit jest tutaj całkowicie odmienny – osoba która programuje musi opisać krok po kroku budowanie i zarządzanie interfejsem graficznym (tak zwane podejście imperatywne) . Z tego też tytułu wynika sporo różnic pomiędzy tymi frameworkami

1) UIKit zachowuje z tego tytułu pełną kontrolę nad tym czym zajmuje się użytkownik (interfejs) natomiast SwiftUI ma mocno uproszczoną składnię (łatwiej zlokalizować błędy i poprawić ewentualne fragmenty kodu)

2) Druga kwestia to sam kod - ten w SwiftUI ma dużo wbudowanych funkcji (motywy które możemy deklarować , lokalizacja , edycja czcionki w wielu zakresach itd.) natomiast ten zawarty w UIKit jest zdecydowanie bardziej obszerny , trudniejszy w utrzymaniu i ciężki do zrozumienia przez początkowych programistów

3) Kolejna kwestia to wsparcie – UIKit oferuje wsparcie dla Objective C oraz Swifta natomiast SwiftUI kreuje „Podgląd na żywo”

4) Ze względu na to , że UIKit jest dużo dłużej dostępny na rynku ma takie zalety jak: Dużo zasobów i baz wiedzy i dostępny jest na IOS-a już od 9 wersji włącznie. Natomiast analogicznie SwiftUI przez mniejszy bagaż czasowy ma mniej zasobów z których możemy korzystać oraz dostępny jest dopiero od 13 wersji IOS-a

Zadanie 3

Pytanie: Czym jest typ opcjonalny w języku Swift? Wymień i uzasadnij jego przykładowe zastosowania

Typ opcjonalny w języku Swift, jak sama nazwa mówi, to mechanizm, który pozwala zmiennej przechowywać albo konkretną wartość, albo jej brak (nil) . Dzięki temu możemy bezpiecznie pracować z danymi, które mogą, ale nie muszą istnieć w danym momencie działania programu.

Łatwiej zrozumieć to na przykładzie, w którym jakiś wynik może nie zostać znaleziony. Dobrym przypadkiem jest funkcja wyszukująca użytkownika w bazie danych – jeżeli użytkownik istnieje, funkcja zwróci jego dane, a jeśli nie – zwróci nil, czyli brak wyniku.

Aby jeszcze łatwiej można było to zrozumieć przykład na liczbach również będzie świetnym wpisaniem się w definicję typu opcjonalnego. Chodzi oczywiście o dzielnie przez zero – jak wiemy jest to niedozwolone z punktu matematycznego. Aby temu zapobiec, możemy stworzyć funkcję, która przyjmuje licznik i mianownik, a następnie sprawdza, czy mianownik jest różny od zera.

Jeśli tak — zwracany jest wynik dzielenia. Jeśli nie — funkcja zwraca nil, czyli informację, że wynik nie istnieje, bo dzielenie nie może zostać wykonane. W takim przypadku typ zwracany przez funkcję musi być opcjonalny, ponieważ nie zawsze jesteśmy w stanie zwrócić wartość liczbową.

Zadanie 4

Pytanie: Czym jest domknięcie w języku Swift? Wymień i uzasadnij jego przykładowe zastosowania

Domknięcie w języku Swift to specjalny typ bloku kodu, który można przypisać do zmiennej lub przekazać jako argument do funkcji.

Najprościej mówiąc - jest to fragment kodu, który można „zapamiętać" i wykonać później. Przypomina funkcję, ale może być anonimowe (czyli nie musi mieć nazwy).

Podajmy to standardowo jak w poprzednim zadaniu na przykładzie (będzie to łatwiej zrozumieć) → wyobraźmy sobie że pobieramy jakieś dane z internetu z jakiegoś serwera (niech to będzie lista produktów spożywczych jakiegoś sklepu dużej marki).

Proces może chwilę potrwać więc możemy zrobić coś takiego – zamiast czekać bezczynnie przekazujemy domknięcie jako argument czyli mówimy , że gdy skończysz pobierać dane X to wykonaj kawałek kodu Y. Tym samym zachowujemy logikę domknięcia czyli wykonujemy coś później po zakończeniu jakiegoś zadania.

Zadanie 5

Pytanie: Scharakteryzuj kolekcje dostępne w języku Swift. Podaj przykładowe zastosowania

Język Swift zawiera trzy podstawowe typy kolekcji: tablice , zbiory oraz słowniki.

Tablica (od ang. Array) - To uporządkowana kolekcja elementów tego samego typu. Każdy element ma konkretny indeks (czyli pozycję) i może się powtarzać. Tablice są idealne, gdy chcemy przechować dane w określonej kolejności i mieć do nich dostęp przez numer pozycji.

Przykład na którym możemy to pokazać: Mamy listę zakupów z której chcemy kupić chleb , mleko i jajka. Istotna jest kolejność – jako pierwszy weźmiemy chleb potem mleko a następnie jajka (tablica pozwala przechowywać tą listę zakupów w takiej kolejności).

Umożliwia nam również poprzez indeks odczytać co jest na pierwszym miejscu co na drugim itd.

I co również ważne i opisaliśmy w definicji powyżej elementy mogą się powtarzać (czyli na przykład mleko możemy kupić dwa razy jeśli interesują nas dwie butelki)

Zbiór (od ang. Set) – To z kolei nieuporządkowana kolekcja unikalnych elementów. Elementy nie mają ustalonej kolejności i nie mogą się powtarzać. Zbiory są szybkie w sprawdzaniu, czy dana wartość istnieje w kolekcji, i przydają się, gdy nie interesuje nas kolejność, tylko unikalność danych.

Przykład na którym możemy to pokazać: Aby łatwiej było to zobaczyć weźmiemy dokładnie tą samą listę zakupów. Zmienimy tylko jej warunki po zbiór oraz słownik w następnym akapicie.

W zbiorze te produkty zgodnie z jego definicją wyświetlą się jako unikalne elementy bez informacji o ilości oraz bez określonej kolejności (jak miało to miejsce w przypadku Tablicy). Wizualnie wyglądałoby to tak , że mamy chleb , jajka oraz mleko i nie wiemy który z elementów kupimy pierwszy i w jakiej ilości.

Słownik (od ang. Dictionary) - To kolekcja przechowująca dane w postaci par klucz-wartość. Każdy klucz musi być unikalny, natomiast wartości mogą się powtarzać. Słowniki świetnie nadają się do szybkiego wyszukiwania informacji, gdy znany jest klucz.

Przykład zastosowania (jak opisany wyżej ten sam) → nasza lista będzie wyglądać teraz tak że produkt będzie kluczem a wartość to ilość jaką chcemy kupić:

Chleb: 1

Mleko: 2

Jajka: 12

Itd.

Zgodnie z założeniami możemy mieć na przykład w tej liście Jajka : 12 oraz Mleko : 12 (wartości mogą się powtarzać) ale nie możemy już mieć Jajka : 6 oraz Jajka : 12 (klucze muszą mieć wartość unikalną)

Zadanie 6

Pytanie: Opisz różnice pomiędzy klasą generyczną , funkcją generyczną i strukturą w języku Swift. Podaj przykładowe zastosowania

Zgodnie z poleceniem wypiszemy definicje tych elementów , podstawowe różnice oraz przykładowe zastosowanie (tym razem posłużymy się innymi przykładami aby lepiej to zobrazować)

Klasa generyczna – specjalny rodzaj klasy, która jest definiowana z parametrem typu, co oznacza, że może działać na różnych typach danych bez konieczności pisania osobnej klasy dla każdego z nich.

Dzięki temu klasa generyczna jest bardzo elastyczna i wielokrotnego użytku. Jako typ referencyjny, klasa przechowuje swoje dane w jednym miejscu w pamięci, a kiedy przekazujemy obiekt klasy, operujemy na tej samej instancji.

Przykład: Mamy magazyn do zarządzania zasobami firmy. Ten magazyn może przechowywać różne rodzaje przedmiotów — komputery, meble biurowe dokumenty itd. Dzięki klasie generycznej nie musimy tworzyć osobnej klasy magazynu dla każdego

typu rzeczy — jedna klasa działa dla wszystkich rodzajów przedmiotów, bo jest uniwersalna i elastyczna.

Funkcja generyczna – to funkcja, która również korzysta z parametrów typu, co pozwala na operowanie na różnych typach danych w ramach jednej implementacji funkcji. To znaczy, że podczas wywołania funkcji możemy zdecydować, jaki konkretny typ danych ma zostać użyty, bez konieczności pisania oddzielnych funkcji dla każdego typu.

Przykład: Wyobraźmy sobie, że mamy dwie kartki papieru z różnymi notatkami i chcemy je zamienić miejscami na biurku. Możemy napisać funkcję, która zamienia miejscami dowolne dwa przedmioty — nie tylko kartki, ale też długopisy, książki czy inne rzeczy. Ta funkcja działa dla różnych typów przedmiotów, ponieważ jest generyczna i uniwersalna.

Struktura - to z kolei typ wartościowy, który przechowuje dane i metody, podobnie jak klasa, ale różni się tym, że jest przekazywana przez kopiowanie, a nie przez referencję**.** Oznacza to, że kiedy przypisujemy strukturę do nowej zmiennej lub przekazujemy ją do funkcji, tworzona jest jej kopia.

Przykład: Tworzymy aplikację do rysowania i chcemy przechować punkty na płótnie — czyli miejsce, gdzie postawimy kropkę. Struktura to prosty pojemnik na współrzędne x i y tego punktu. Gdy skopiujemy ten punkt, dostaniemy niezależną kopię, więc zmiany w jednym punkcie nie zmienią drugiego.

Zadanie 7

Pytanie: Czym jest Storyboard , Interface Builder i Outlet w UIKit?

Poszczególne definicje:

Storyboard - w UIKit to plik, który pomaga tworzyć wizualny projekt interfejsu aplikacji na iOS. Możemy w nim „układać” ekrany (zwane widokami) i łączyć je ze sobą, pokazując jak użytkownik przechodzi z jednego ekranu do drugiego.

Outlet to specjalne połączenie między elementem interfejsu w storyboardzie (na przykład przyciskiem, etykietą czy obrazem) a kodem w programie. Dzięki outletowi możesz w kodzie zmieniać właściwości tych elementów, na przykład zmienić tekst na przycisku albo ukryć jakiś obraz.

Interface Builder to narzędzie , które umożliwia projektowanie interfejsu graficznego za pomocą przeciągania i upuszczania elementów na ekran. To właśnie w Interface Builderze tworzymy storyboardy i łączymy elementy z kodem za pomocą outletów (wspomnianych wyżej) i akcji.

Zadanie 8

Pytanie: Za co odpowiada mechanizm AutoLayout? Opisz ogólną zasadę działania. Do czego można porównać działanie takiego mechanizmu?

Mechanizm Auto Layout w UIKit odpowiada za automatyczne ustawianie i dostosowywanie elementów interfejsu użytkownika na ekranie w sposób elastyczny i zgodny z różnymi rozmiarami urządzeń oraz orientacjami ekranu.

Dzięki Auto Layout nie musimy ręcznie ustawiać pozycji i rozmiarów każdego elementu dla każdej możliwej wielkości ekranu — system sam dba o to, aby interfejs wyglądał dobrze zarówno na małym iPhonie, jak i dużym iPadzie.

Zasada działania AutoLayouta polega na tak zwanych ograniczeniach między elementami interfejsu lub względem samego ekranu. Na przykład można powiedzieć, że przycisk ma być zawsze wyśrodkowany, albo że etykieta ma znajdować się 20 punktów poniżej innego widoku. Auto Layout używa tych ograniczeń, aby wyliczyć optymalne rozmiary i pozycje elementów, dostosowując się do różnych warunków.

Do czego możemy to porównać? Najtrafniejszy przykład z życia codziennego to podejrzewam układanie puzzli lub budowania konstrukcji z klocków LEGO, gdzie każde połączenie (ograniczenie) mówi, jak elementy mają do siebie pasować.

Zadanie 9

Pytanie: Czym są niespełnione ograniczenia i niejednoznaczne pozycjonowanie? Podaj przykłady wystąpień.

Obie te funkcjonalności odnoszą się do AutoLayoutu omawianego również w poprzednim pytaniu

Niespełnione ograniczenia pojawiają się wtedy, gdy system Auto Layout nie jest w stanie jednocześnie spełnić wszystkich nałożonych na elementy interfejsu ograniczeń.

Może się to zdarzyć na przykład, gdy ustawimy, że przycisk ma mieć jednocześnie określoną szerokość i być przyklejony do lewej i prawej krawędzi ekranu, ale przestrzeń jest za mała, by to zrealizować (czyli pojawia się problem definiowany jako niespełnione ograniczenia)

Niejednoznaczne pozycjonowanie występuje, gdy Auto Layout nie ma wystarczająco informacji, by jednoznacznie określić pozycję lub rozmiar elementu.

Na przykład, jeśli nie zdefiniujemy, gdzie dokładnie ma się znajdować widok ani jaką ma mieć szerokość czy wysokość, system nie wie, jak go ustawić.

Zadanie 10

Pytanie: Czym są priorytety ograniczeń w AutoLayout? Podaj przykłady zastosowań

Priorytety ograniczeń w Auto Layout to wartości, które określają, jak ważne jest dla systemu spełnienie danego ograniczenia podczas układania elementów interfejsu.

Każde ograniczenie ma przypisany priorytet w skali od 1 do 1000, gdzie 1000 oznacza obowiązkowe (musi być spełnione) a wartości poniżej 1000 to ograniczenia opcjonalne, które system może złamać, jeśli zajdzie taka potrzeba, by zachować spójność całego układu.

Na tapetę możemy wziąć tutaj przykład z marginesami , które chcemy ustawić: Możemy chcieć, żeby margines z lewej strony był zawsze zachowany, ale margines z prawej może być mniejszy, jeśli brakuje miejsca. Wtedy lewy margines dostanie wyższy priorytet niż prawy (co idealnie wpisuje się w priorytety ograniczeń w AutoLayoucie)