Kurs programowania w Swift - notatki

Autor: Szymon Maćkowiak

Literatura:

1. Podręcznik Apple: "The Swift Programming Language - Swift 5.5 Edition"
2. Strona dokumentacji jezyka Swift: https://docs.swift.org
3. Strona z zasobami Apple dla programistów: https://developer.apple.com

Co to jest Swift?

Swift to współczesny język programowania opracowany przez firmę Apple. Umożliwia tworzenie oprogramowania na cały ekosystem urządzeń firmy Apple (iPhone, Mac, iPad, watch). Jednak programy napisane w języku Swift mogą byc wykorzystywane w innych systemach, a nawet na serwerach. Swift uchodzi za język bezpieczny i zoptymalizowany pod kątem wydajności i zastosowań biznesowych. Równocześnie jest przyjazny dla nowych programistów.

Najważniejsze cechy języka (dyktowane praktycznymi względami) to:

- zmienne są zawsze inicjalizowane przed wykorzystaniem,
- indeksy tablic są sprawdzane pod kątem wykraczania poza ich zakres,
- zmienne typu całkowitego są sprawdzane pod kątem przekroczenia zakresu,
- opcje (optionals) zapewniają, że wartości zerowe (nil - pusty zbiór lub lista) są obsługiwane wprost (explicite),
- pamięć jest zarządzana automatycznie,
- obsługa błędów pozwala na unikanie nieoczekiwanych awarii.

Zadanie 1.

Otwórz program Xcode i utwórz nowy projekt (Create a new Xcode project). Następnie zaznacz kartę macOS i wybierz opcję "Command Line Tool". W polu "Product Name" nadaj nazwę swojemu programowi, upewnij się, że w polu "Language" masz wybrany język programowania Swift i naciśnij przycisk "Next". Wybierz miejsce, w którym ma zostać zapisany i naciśnij przycisk "Create". W obszarze drzewa projektu po lewej stronie wybierz (przez kliknięcie) plik "main". Zmodyfikuj kod pliku w taki sposób, aby przywitał się ze światem w Twoim ojczystym języku. Uruchom program za pomocą przycisku strzałki i poczekaj na wyświetlenie wyniku na konsoli w dolnej części edytora.

Przykładowe rozwiązanie:

```
import Foundation

print("Witaj Świecie!")
```

Tym sposobem dochowaliśmy tradycji i napisaliśmy nasz pierwszy program w języku Swift - tzw. "Hello, World!".

Komentowanie

Ważnym elementem każdego języka programowania jest możliwość komentowania kodu. W języku Swift możemy stosować komentarze jednoliniowe (//) oraz wieloliniowe (/* ... */). Przykład zastosowania komentarzy możesz zauważyć spoglądając na automatycznie wygenerowany nagłówek w Twoim pierwszym programie.

Tworzenie zmiennych

Przez zmienną będziemy rozumieć "słowo, do którego zapisane zostały pewne dane". W języku Swift mamy rozróżnienie na zmienne i stałe. Słowo kluczowe let służy do tworzenia stałych, a var do tworzenia zmiennych. Wartość stałej nie musi być znana w momencie kompilacji, ale należy dokonać przypisania do niej tylko raz (możemy z niej korzystać w wielu miejscach programu, ale przypisania wartości dokonujemy tylko raz).

```
var mojaZmienna = 12
mojaZmienna = 20
let mojaStala = 13
```

Stała musi mieć ten sam typ co dane do niej wprowadzane, aczkolwiek nie zawsze trzeba wpisywać to wprost (explicite). Kompilator może sam się domyślić jakiego typu ma być stała (implicite, domyślnie). Jeżeli chcemy narzucić typ, możemy to zrobić wpisując po nazwie zmiennej (po dwukropku) żądany typ.

```
let domyslnaCalkowita = 70
let domyslnaZmiennoprzecinkowa = 70.0
let wyraznaZmiennoprzecinkowa: Double = 70
```

Zmianę typu możemy dokonać poprzedzając nazwę zmiennej nazwą żądanego typu (Int(), Double(), String(), itd.).

Zadanie 2.

W środowisku Xcode utwórz zmienną i stałą. Spróbuj zmienić wartość stałej. Wyniki wyświetl w konsoli.

Przykładowe rozwiązanie:

```
import Foundation

var myVariable: Double = 42
let zm2 = 50

myVariable = 43
```

```
print("Hello, World!", myVariable, zm2)
```

Zadanie 3

Utwórz dwie dowolne zmienne i wyświetl w konsoli wynik następujących operacji (arytmetycznych): +, -, *, /, %. Nazwij operacje realizowane przez kolejne operatory.

Przykładowe rozwiązanie

```
import Foundation

var myFirstVriable = 42
let mySecondVriable = 5

print(myFirstVriable + mySecondVriable) //
print(myFirstVriable - mySecondVriable)
print(myFirstVriable * mySecondVriable)
print(myFirstVriable / mySecondVriable)
print(myFirstVriable % mySecondVriable)
```

Zadanie 4

Utwórz dwie dowolne zmienne i wyświetl w konsoli wynik następujących operacji (logicznych): ==, !=, >, <, >=, <=. Nazwij operacje realizowane przez kolejne operatory.

Przykładowe rozwiązanie

```
import Foundation

var myFirstVriable = 5
let mySecondVriable = 5

print(myFirstVriable == mySecondVriable) //
print(myFirstVriable != mySecondVriable)
print(myFirstVriable > mySecondVriable)
print(myFirstVriable < mySecondVriable)
print(myFirstVriable >= mySecondVriable)
print(myFirstVriable <= mysecondvriable) < pre>
```

Wartości nigdy nie są domyślnie konwertowane na inny typ. Jeżeli musimy dokonać konwersji na inny typ (rzutowanie), należy ustalić wyraźnie (explicite) żądany typ:

```
let label = "Długość wynosi "
let length = 94
let length_label = label + String (length)
```

Istnieje prostszy sposób na umieszczanie wartości w łańcuchu znaków - możemy wewnątrz łańcucha znaków użyć operatora "\()". Wewnątrz opertora "\()" można wykonywać również obliczenia na zmiennych. Z kolei operator potrójnego cudzysłowu """ umożliwia tworzenie wieloliniowych łańcuchów znaków.

Zadanie 5 Utwórz zmienną przechowującą aktualny rok, a następnie wyświetl komunikat o aktualnym roku z wykorzystaniem operatora \().

Przykładowe rozwiązanie

```
let year = 2022
print("Aktualnie trwa rok \(year)")
```

Zadanie 6 Utwórz dwie zmienne przechowujące zawartość koszyka (5 jabłek i 3 banany), a następnie (korzystając z operatora ()) wyświetl zdanie "W moim koszyku są X jabłka i Y banany." (gdzie X i Y to odpowiednie dane).

Przykładowe rozwiązanie

```
let apples = 5
let bananas = 3
print("W moim koszyku są \(apples) jabłka i \(bananas) banany.")
```

Do tworzenia list, tablic i słowników służy operator nawiasu kwadratowego [ ]. Dostęp do elementów listy odbywa się przez indeks, a do elementów słownika przez klucz:

```
var myList = ["apples", "bananas", "potatoes"]
myList[1] = "oranges"
```

```
var romanNumbersDictionary = [
    "I" : 1,
    "II" : 2,
    "X" : 10,
]
romanNumbersDictionary["V"] = 5
```

Kolejne elementy do list możemy dodawać korzystając z metody append. Listy automatycznie rosną w miarę dodawania do nich elementów.

```
myList.append("bread")
print(myList)
```

Tworzenie pustych list umożliwia składnia inicjalizacyjna:

```
let myEmptyArray: [Int] = []
let myEmptyDictionary: [String: Int] = [:]
```

Jeżeli typ danych może być wywnioskowany (np. w sytuacjach: gdy ustawia się nową wartość dla zmiennej lub przekazuje argument do funkcji) puste listy i słowniki można tworzyć korzystając z operatora [] i odpowiednio [:].

```
myList = []
romanNumbersDictionary = [:]
```

Sterowanie przepływem

W języku Swift do kontroli przepływu w programie możemy wykorzystać instrukcje warunkowe (if) lub przełączniku (switch). Z kolei w kontekście pętli mamy do dyspozycji następujące (for-in, while, repeat-while). Nawiasy wokół warunków lub zmiennych pętli są opcjonalne. Klamry wokół ciał warunków i pętli są obligatoryjne.

```
let studentScores = [23, 54, 88, 12]
var passingExam = 0

for score in studentScores {
    if score > 50 {
        passingExam = 1
    } else {
        passingExam = 0
    }

print("Liczba punktów \(score), czy zdane? \(passingExam)")
}
```

Warunek w instrukcji if musi być wyrażeniem logicznym. Można korzystać z klauzuli if i let razem do pracy z brakującymi wartościami. Te wartości są reprezentowane jako opcjonalne. Wartość opcjonalna zawiera albo wartość albo obiekt nil aby wskazać, że danej wartości brakuje. Aby skorzystać z typu opcjonalnego należy użyć znaku zapytania (?) po typie wartości.

Uzupełnienie: w języku Swift nil oznacza brak wartości. Przesłanie danych do nil skutkuje błędem programu.

Porównaj działanie poniższych programów:

```
var myOptionalName: String?
print(myOptionalName == nil)
```

Powyższy program zwraca wartość true.

```
var myOptionalName: String? = "My Name"
print(myOptionalName == nil)
```

Powyższy program zwraca wartość false.

```
var myOptionalName: String? = nil
print(myOptionalName == nil)
```

Powyższy program zwraca wartość true.

Jeżeli opcjonalna wartość w warunku jest równa nil, wartość logiczna warunku wynosi false, a kod bloku warunkowego zostanie pominięty.

Wartość opcjonalną można zastąpić wartością domyślną. Umożliwia to operator podwójnego znaku zapytania (??). Aby przypisać do wartości opcjonalnej wartość domyślną, należy po jej nazwie użyć operatora ?? a za nim wartość bądź nazwę zmiennej, która ma zostać użyta jako domyślna.

```
let nick: String? = nil
let name: String = "Nowak"
print("Hello \(nick ?? name)")
```

Kolejną konstrukcją, która umożliwia kontrolę przepływu w programie jest instrukcja switch. Wspiera ona każdy rodzaj danych oraz szeroką różnorodność operacji porównania.

```
let fruit = "strawberry"

switch fruit{

    case "banana":
    print("banana is yellow")

    case "apple":
    print("apple is red")

    case "orange":
```

```
    print("oragne is orange")

default:
    print("I don't know the color")

}
```

Usunięcie klauzuli default spowoduje wystąpienie błędu.

Wejście do pasującego bloku i jego wykonanie w konstrukcji switch spowoduje wyjście poza blok i przejście do kolejnych części programu. Kolejne części bloku switch nie są sprawdzane ani wykonywane. Nie ma potrzeby wstawiania instrukcji break (jak np. w językach C / C++).

Pętle

Pętle służą do wykonywania fragmentu programu więcej niż raz. Jeżeli interesuje nas pętla wykonywana po uporządkowanej sekwencji, możemy skorzystać z pętli for:

```
for i in 1...10 {
    print(i)
}
```

Warto zwrócić uwagę, że w języku Swift wykonywany jest zarówno lewy i prawy zakres licznika pętli.

Aby wykonać iterację po elementach słownika można skorzystać z pętli for-in:

```
let romanNumbers = [
    "I": 1,
    "II": 2,
    "V": 5
]

for (i, j) in romanNumbers{
    print(i, j)
}
```

Kolejną istotną pętlą, jest pętla while. Nie posiada ona wbudowanego licznika. Jest wykonywana o ile spełniony jest jej warunek (logiczny).

```
var counter = 0
while (counter<20){ print(counter) counter+="1" } < pre>
```

Istnieje również odpowiednik do konstrukcji do...while znanych z języków C/C++ i nosi ona nazwę repeat:

```
var m = 0
repeat{
    print(m)
    m += 1
} while m < 10
```

W języku Swift instrukcje sterujące i pętle mogą być wzbogacone dodatkowo o instrukcje:

- break
- continue
- fallthrough
- return
- throw