LABORATORIUM 5. INSTRUKCJE STERUJĄCE.

Cel laboratorium:

Poznanie instrukcji sterujących w języku Swift.

Zakres tematyczny zajęć:

- instrukcja sterująca *while*,
- instrukcja sterująca *repeat-while,*
- instrukcja sterująca *for-in*,
- instrukcje: *continue* oraz *break*.

Pytania kontrolne:

1. Jakie dostępne są instrukcje sterujące w języku Swift?
2. W jaki sposób działa instrukcja *while*?
3. W jaki sposób działa instrukcja *repeat-while*? Kiedy powinno się ją stosować?
4. Kiedy można stosować instrukcję *for-in*?
5. W jaki sposób można w instrukcji for-in przechodzić po elementach z różnym krokiem?
6. W jaki sposób działa instrukcja *continue*?
7. W jaki sposób działa instrukcja *break*?

W języku Swift dostępne są instrukcje sterujące w postaci instrukcji iteracyjnych (pętli) takich jak: *while* oraz *repeat-while* do wielokrotnego wykonania bloku instrukcji, czy for-in, stosowanej m.in. do iteracji tablic, słowników, czy łańcuchów.

Instrukcja *while* w pierwszej kolejności sprawdza podany logiczny warunek. Jeśli jest on prawdziwy (ma wartość *true*), wykonywane są instrukcje zawarte w pętli. Jeśli warunek jest fałszywy (*false*), instrukcja jest kończona. Instrukcja ta może nigdy nie zostać wykonana, jeśli warunek będzie fałszywy. Należy także pamiętać, aby zapewnić, by warunek zmienił wartość na *false*. W przeciwnym razie pętla będzie nieskończona.

Na Listingu 5.1 przedstawiono wyświetlenie nieparzystych liczb całkowitych od 1 do 21. Należy pamiętać o ustawieniu wartości początkowych, a także ich zmianę w kolejnych iteracjach pętli.

Listing 5.1. Wyświetlenie liczb nieparzystych z zastosowaniem instrukcji while

```
var a = 1

while a <= 21{
   print(a)
    a+=2
}
```

Instrukcja *while* stosowana jest także w przypadku, gdy nie jest znana liczba iteracji. Na Listingu 5.2 przedstawiono obliczenie sumy losowych liczb (z zakresu od 1 do 9). Liczby są losowane do chwili, gdy ich suma osiągnie co najmniej 50.

Listing 5.2. Obliczenie sumy liczb losowych z zastosowaniem instrukcji while

```
var a: Int
var sum = 0

while sum <= 50{
    a = Int.random(in: 1..<10)
    sum += a
    print("Wylosowano liczbę \(a)")
}

print("Suma wszystkich elementów wynosi \(sum)")
```

Instrukcja *repeat-while* jest pętlą, w której najpierw wykonywane są instrukcje, a następnie sprawdzany jest warunek. Oznacza to, że ta instrukcja zostanie wykonana przynajmniej raz. Jeśli podany warunek jest prawdziwy, wykonywane są kolejne iteracje. W przeciwnym wypadku instrukcja jest kończona.

Na listingu 5.3 przedstawiono wyświetlenie liczb nieparzystych od 1 do 21 z użyciem instrukcji *repeat-while*, a na Listingu 5.4 obliczenie sumy elementów do momentu uzyskania przynajmniej 50.

Listing 5.3. Wyświetlenie liczb nieparzystych z zastosowaniem instrukcji repeat-while

```
var a = 1

repeat{
    print(a)
     a+=2
}while a <= 21
```

Instrukcję tę można także zastosować do sprawdzenia wprowadzanych danych. Przykład weryfikacji, czy wczytana liczba jest dwucyfrowa, został przedstawiony na Listingu 5.5. Wczytany łańcuch jest konwertowany na liczbę. Jeśli operacja się powiedzie, zmienna *a* będzie zawierała liczbę, a w przeciwnym razie będzie pusta (*nil*). Dodatkowo należy sprawdzić zakres liczby, czy jest dwucyfrowa. Z tego powodu w warunku są sprawdzane te dwie ewentualności. Jeśli liczba nie może zostać przekonwertowana na liczbę lub nie jest ona dwucyfrowa, zostanie ponownie wyświetlony tekst proszący o podanie liczby dwucyfrowej.

Listing 5.4. Obliczenie sumy liczb losowych z zastosowaniem instrukcji repeat-while

```
var a: Int
var sum = 0

repeat{
    a = Int.random(in: 1..<10)
    sum += a
    print("Wylosowano liczbę \(a)")
}
while sum <= 50

print("Suma wszytkich elementów wynosi \(sum)”)
```

Listing 5.5. Sprawdzenie, czy wczytana liczba jest dwucyfrowa

```
var str: String?
var a: Int?

repeat{
    print("Podaj liczbę dwucyfrową: ")
    str = readLine()
    a = Int(str!)
} while(a == nil) || ( (a! < 10) || (a! > 99))
```

Instrukcję sterującą *for-in* stosuje się, gdy znana jest liczba iteracji. Na Listingu 5.6 przedstawiono wyświetlenie kolejnych liczb całkowitych od 1 do 20 z użyciem zamkniętego operatora (ang. *Closed Range Operator*). Instrukcja ta sama przechodzi po kolejnych elementach kolekcji. Nie można modyfikować wartości zmiennej iteracyjnej.

Listing 5.6. Wyświetlenie kolejnych liczb z zakresu 1-20 z zastosowaniem instrukcji for-in

```
for i in 1...20 {
    print(i)
}
```

Instrukcja *for-in* umożliwia także przejście po elementach z założonym krokiem. W tym celu należy skorzystać z funkcji *stride(from:to:by:)*, która zwraca sekwencję od wartości początkowej (*from*) do wartości końcowej (*to*), ale bez niej, z krokiem o określonej wartości (*by*). Przykład wyświetlenia liczb z zakresu od 1 do 30 z krokiem 3 oraz obliczenia ich sumy został przedstawiony na Listingu 5.7.

Listing 5.7. Obliczenie sumy liczb z zakresu 1-30 z krokiem 3 z zastosowaniem instrukcji for-in

```
var sum = 0
for i in stride(from: 0, to: 31, by: 3){
    print(i)
    sum+=i
}

print("Suma elementów wynosi \(sum)”)
```

Przy użyciu funkcji *stride(from:to:by:)* można wygenerować elementy z krokiem malejącym, co przedstawiono na Listingu 5.8.

Listing 5.8. Wyświetlenie liczb od 20 do 0 z krokiem -5 z zastosowaniem instrukcji for-in

```
for i in stride(from: 20, to: -1, by: -5){
    print(i)
}
```

Instrukcje sterowania zmieniają kolejność wykonywania kodu, przenosząc kontrolę z jednego fragmentu kodu do drugiego. W implementacji instrukcji sterujących stosowane są często:

- continue;
- break.

Instrukcja *continue* powoduje, że dana iteracja pętli jest przerywana, a kod do końca instrukcji jest pomijany. Następnie program przechodzi do kolejnej iteracji pętli. Na Listingu 5.9 przedstawiono obliczenie sumy elementów z zakresu od 5 do 20 z pominięciem liczb, które podzielne są przez 7. Jeśli warunek instrukcji *if* jest prawdziwy, wykonywanie danej iteracji jest przerywane i następuje przejście do kolejnej.

Listing 5.9. Suma liczb z zakresu od 5 do 20 z pominięciem liczb podzielnych przez 7

```
var sum = 0
for i in 5...20{
    if i % 7 == 0 {
        continue
    }
    sum += i
}

print("Suma elementów wynosi \(sum)”)
```

Na Listingu 5.10 przedstawiono wyświetlenie cyfr od 1 do 7 z pominięciem liczb parzystych. Instrukcję *continue* została użyta w instrukcji *while*.

Listing 5.10. Wyświetlenie cyfr od 1 do 7 z pominięciem liczb parzystych

```
var i = 1
while i <= 7{
    i += 1
    if i % 2 == 0 {
        continue
    }
    print(i)
}
```

Instrukcja *break* natychmiast kończy wykonywanie całej instrukcji sterowania. Instrukcja *break* może być używana wewnątrz pętli, jeśli musi nastąpić zakończenie instrukcji wcześniej, niż wynika to z warunku. Nie wykonywany jest żaden kod znajdujący się w pętli.

Przykład instrukcji *break* został przedstawiony na Listingu 5.11. Pętla kończy się, jeśli użytkownik wpisze znak *a*. Wewnątrz pętli sprawdzane jest, czy wczytany znak jest równy *z*. Jeśli warunek jest prawdziwy, pętla także wtedy kończy działanie.

Listing 5.11. Przerwanie instrukcji repeat-while

```
var str: String?
repeat{
    print("Podaj dowolny znak: ")
    str = readLine()
    if str == "z" {
        break
    }
    print("Wpisałeś \(str!)")

}while str != "a"
```

Zadanie 5.1. Utwórz aplikację konsolową – ciąg Fibonacciego

Polecenie 1. Napisz program, który wczyta pewną liczbę do zmiennej *licz* oraz wyliczy i wyświetli kolejne elementy ciągu Fibonacciego, które nie przekraczają wartości wczytanej zmiennej. W przypadku, gdy kolejny wyliczony element jest większy od zmiennej licz, program powinien zaprzestać wyświetlania. Zastosuj instrukcję *while*.

Zadanie 5.2. Utwórz aplikację konsolową – liczba pierwsza

Polecenie 1. Napisz program, który wczyta pewną liczbę, a następnie sprawdzi, czy jest to liczba pierwsza.

Zadanie 5.3. Utwórz aplikację konsolową – średnia cyfr

Polecenie 1. Napisz program, który wczyta **liczbę** i obliczy średnią geometryczną cyfr. Należy wymusić poprawność wczytanej liczby, aby była ona co najmniej trzycyfrowa za pomocą instrukcji *for*. Należy upewnić się, że wczytany ciąg znaków jest liczbą.

Zadanie 5.4. Utwórz aplikację konsolową – palindrom

Polecenie 1. Napisz program, który sprawdzi, czy wczytany ciąg znaków jest palindromem (czytany od prawej lub lewej strony daje taki sam łańcuch). Należy zastosować pętlę!

Zadanie 5.5. Utwórz aplikację konsolową – minima i maksima lokalne

Polecenie 1. Napisz program, który wczyta ciąg liczb całkowitych $\{a_i\}$, i=1, 2, …, n. Należy zapewnić, że $a_i \neq a_{i+1}$podczas wczytywania danych. Należy znaleźć i wyświetlić liczbę minimów i maksimów lokalnych w tym ciągu, przesuwając się z krokiem 1.
Przykład:
a= {1, 2, 0, 3, 8, 7}

1. (1, 2, 0) → max = 1, min = 0
2. (2, 0, 3) → max = 1, min = 1
3. (0, 3, 8) → max = 1, min = 1, bo ciąg rosnący
4. (3, 8, 7) → **max = 2, min = 1**