LABORATORIUM 9. ZŁOŻONE TYPY DANYCH: STRUKTURY, ZBIORY, SŁOWNIKI.

Cel laboratorium:

Poznanie struktur, zbiorów i słowników oraz posługiwanie się nimi w języku Swift.

Zakres tematyczny zajęć:

- tworzenie struktur,
- tworzenie zbiorów,
- wykonywanie operacji na zbiorach,
- tworzenie słowników,
- wykonywanie operacji na słownikach.

Pytania kontrolne:

1. Co to jest struktura? Jak się ją stosuje?
2. Co to jest zbiór? Czy różni się od tablicy?
3. Jak używa się zbiorów?
4. Jakie operacje można wykonywać na zbiorach?
5. Do czego stosuje się słowniki?
6. Jakie operacje można wykonać na słownikach?

Structura (ang. *Structure*) to bloki kodu programu, które zawierają właściwości oraz metody. Definiują nowy typ.

Przykład struktury został przedstawiony na Listingu 9.1. Po słowie kluczowym *struct* podawana jest nazwa struktury, a w nawiasach klamrowych jego składowe. Aby zadeklarować stałą lub zmienną, należy odwołać się do utworzonego typu oraz uzupełnić wartości. Struktury posiadają domyślny inicjalizator.

Listing 9.1. Definiowanie struktur

```
struct Prostopadloscian {
    var a: Double
    var b: Double
    var h:Double
}

struct Osoba {
    var imie: String
    var wiek: Int
}
```

```
let pp = Prostopadloscian(a: 12.0, b: 20, h: 100)

let os = Osoba(imie: "Janek", wiek: 21)
```

Modyfikacja wartości polega na przypisaniu odpowiedniej wartości do zmiennej (Listing 9.2).

Listing 9.2.Modyfikowanie zmiennych

```
var pp = Prostopadloscian(a: 12.0, b: 20, h: 100)

pp.a = 10
pp.b = 25
pp.h = 70
```

Struktura może zawierać inicjalizator oraz metody. Przykład inicjalizacji został przedstawiony na Listingu 9.3. Inicjalizacja może dotyczyć pojedynczych elementów struktury lub wszystkich.

Listing 9.3. Inicjalizacja struktury

```
struct Osoba {
    var imie: String
    var wiek: Int

    init(imie: String, wiek: Int) {
        self.imie = imie
        self.wiek = wiek
    }
}


var o = Osoba(imie: "Andrzej", wiek:35)

print(o)
```

Zbiór (ang. *Sets*) to zestaw, który przechowuje różne wartości tego samego typu w kolekcji bez zdefiniowanej kolejności, w przeciwieństwie do tablicy. Jeśli kolejność elementów nie jest istotna, zamiast tablicy, można użyć zbioru. W zbiorze element o danej wartości może wystąpić tylko raz.

Przykład deklaracji zbiorów został przedstawiony na Listingu 9.4. Pierwszy przechowuje znaki, a drugi łańcuchy. Dodanie nowej wartości do zbioru realizowane są poprzez metodę *insert()*. Aby wyzerować zbiór, wystarczy podpisać znak *[]* lub usunąć wszystkie jego elementy.

Listing 9.4. Deklaracja zbioru

```
var znaki = Set<Character>()

var osoby = Set<String>()

znaki.insert("z") //z
znaki.insert(";") //z, ;
print(znaki)
znaki = [] //pusty zbiór

osoby.insert("Monika")
osoby.insert("Janek”)
```

Zbiór można zdefiniować na podstawie literału tablicowego, co przedstawiono na Listingu 9.5. Zbiór można zadeklarować jawnie podając tym danych lub go pominąć.

Listing 9.5. Deklaracja zbioru z literałem tablicowym

```
var kwiaty: Set<String> = ["róża", "żonkil", "frezja"]

var kwiaty: Set = ["róża", "żonkil", "frezja"]
```

Sprawdzenie, czy zbiór jest pusty, można wykonać na podstawie właściwości *isEmpty*, co przedstawiono na Listingu 9.6. Można także sprawdzić liczbę elementów za pomocą właściwości *count*.

Listing 9.6. Sprawdzenie, czy zbiór jest pusty

```
if kwiaty.isEmpty {
    print("Zbiór jest pusty")
}
else {
    print(kwiaty)
}
```

Dodawanie elementów do zbioru (*insert()*) oraz ich usuwanie (*remove()*) zostało przedstawione na Listingu 9.7. W przypadku sprawdzenia, czy element można usunąć, można sprawdzić, czy on istnieje, co zostało przedstawione na Listingu 9.8. Usunięcie wszystkich elementów zbioru odbywa się przy pomocy metody *removeAll()*.

Listing 9.7. Dodawanie elementów do zbioru

```
kwiaty.insert("fiołek")
print(kwiaty)
//["żonkil", "frezja", "fiołek", "róża"]

kwiaty.remove("róża")
print(kwiaty)
//["żonkil", "frezja", "fiołek"]
```

Listing 9.8. Usuwanie elementów ze sprawdzeniem

```
if let kwiatDoUsuniecia = kwiaty.remove("fiołek") {
    print("Można usunąć \(kwiatDoUsuniecia)")
}
else {
    print("Element do usunięcia nie istnieje")
}
```

Dostęp do poszczególnych elementów zbioru można uzyskać poprzez instrukcję iteracyjną, co zostało przedstawione na Listingu 9.9. Ponieważ zbiór nie ma ustalonej kolejności jego elementów, można zastosować sortowanie elementów (*sorted()*).

Listing 9.9. Dostęp do elementów zbioru

```
var kwiaty: Set = ["róża", "żonkil", "frezja", "fiołek"]

for i in kwiaty {
    print(i)
}

for i in kwiaty.sorted() {
    print(i)
}
```

Na zbiorach można wydajnie wykonywać podstawowe jego operacje za pomocą metod:

- *union()* – suma dwóch zbiorów;
- *intersection()* – część wspólna dwóch zbiorów;
- *symmetricDifference()* – metoda tworzenia nowego zestawu z wartościami w jednym zestawie, ale nie w obu;
- *subtracting()* – odejmowanie zbiorów.

Operacje na zbiorach zostały przedstawione na Listingu 9.10. Każda z metod tworzy nowy zbiór.

Listing 9.10. Operacje na zbiorach

```
var s1: Set = [1, 5, 3, 27,80]
var s2: Set = [2, 4, 27, 80]

var s = s1.intersection(s2).sorted()
print(s) //[27, 80]

s = s1.union(s2).sorted()
print(s) //[1, 2, 3, 4, 5, 27, 80]

s = s1.subtracting(s2).sorted()
print(s) //[1, 3, 5]

s = s1.symmetricDifference(s2).sorted()
print(s) //[1, 2, 3, 4, 5]
```

Na zbiorach można wykonywać także następujące operacje:

- == – sprawdza, czy dwa zbiory są sobie równe;
- *isSubset(of:)* – sprawdza, czy wszystkie wartości zestawu są zawarte w określonym zestawie;
- *isSuperset(of:)* – sprawdza, czy zestaw zawiera wszystkie wartości z określonego zestawu;
- *isStrictSubset(z:)* lub *isStrictSuperset(z:)* – sprawdza, czy zestaw jest podzbiorem, czy nadzbiorem, ale nie jest równy określonemu zestawowi;
- *isDisjoint(with:)* – sprawdza, czy dwa zestawy nie mają wspólnych wartości.

Operacje zostały przedstawione na Listingu 9.11.

Listing 9.11. Operacje na zbiorach

```
if s1.isSubset(of: s2) {
    print("s1 jest podzbiorem s2")
}else {
    print("s1 nie jest podzbiorem s2")
}

if s1.isDisjoint(with: s2) {
    print("s1 i s2 nie maja wspólnych wartości")
} else {
    print("s1 i s2 mają wspólne wartości")
}
```

Słownik (ang. *Dictionary*) przechowuje skojarzenia między kluczami tego samego typu i wartościami tego samego typu w kolekcji, bez zdefiniowanej kolejności. Każda wartość jest powiązana z unikalnym kluczem, który działa jako identyfikator tej wartości w słowniku. Elementy w słowniku nie mają określonej kolejności. Słownik stosuje się w celu wyszukania wartości na podstawie ich identyfikatora.

Definiowanie pustej zmiennej słownikowej zostało przedstawione na Listingu 9.12. Klucz słownika jest liczbą całkowitą, a wartość łańcuchem. Przypisanie wartości następuje do konkretnego klucza.

Listing 9.12. Definiowanie zmiennej słownikowej

```
var sl: [Int: String] = [:] // pusty słownik
print(sl)

sl[1] = "jeden"
sl[21] = "dwadzieścia jeden"
print(sl) //[21: "dwadzieścia jeden", 1: "jeden"]
```

Słownik można utworzyć także na podstawie listy w postaci klucz-wartość, co przedstawiono na Listingu 9.13. Słownik zawiera symbol kraju i jego pełną nazwę. Zliczenie elementów słownika następuje z użyciem właściwości *count*.

Listing 9.13. Definiowanie zmiennej słownikowej

```
var kraje: [String: String] = ["PL": "Polska", "D": "Niemcy",
                               "A": "Austria", "Hr":"Chorwacja"]

print("Zdefiniowano \(kraje.count) elementów")
```

Sprawdzenie, czy słownik jest pusty, wykonuje się poprzez właściwość *isEmpty*.

Listing 9.14. Sprawdzenie, czy słownik jest pusty

```
if kraje.isEmpty {
    print("Słownik jest pusty")
} else {
    print("Słownik zawiera \(kraje.count) elementów")
}
```

Dodawanie elementu do słownika polega na przypisaniu nowej wartości do nowego klucza (Listing 9.15).

Listing 9.15.Dodawanie elementów do słownika

```
kraje["Sk"] = "Słowacja" //dodanie elementu

print(kraje)
//["D":  "Niemcy",  "PL":  "Polska",  "A":  "Austria",  "Hr":
   "Chorwacja", "Sk": "Słowacja"]

kraje["Cz"] = "Czech"
print(kraje)
//["D":  "Niemcy",  "Hr":  "Chorwacja",  "Cz":  "Czech",  "Sk":
   "Słowacja", "PL": "Polska", "A": "Austria"]
```

Modyfikacje elementu słownika polega na przypisaniu wartości dla istniejącego klucza lub zastosowanie metody *updateValue(_:forKey:)* (Listing 9.16). Aby zweryfikować, czy istnieje modyfikowany element, zastosowano instrukcję *if*.

Listing 9.16. Modyfikowanie elementów słownika

```
kraje["Cz"] = "Czechy"
print(kraje)
//["D": "Niemcy", "A": "Austria", "Sk": "Słowacja", "PL": "Polska",
"Hr": "Chorwacja", "Cz": "Czechy"]

//albo
if let kr = kraje.updateValue("Czechy", forKey: "Cz") {
    print("Zmodyfikowano \(kr)")
} else {
    print("Brakuje szukanego elementu")
}
print(kraje)
//["Sk": "Słowacja", "PL": "Polska", "A": "Austria", "D": "Niemcy",
"Hr": "Chorwacja", "Cz": "Czechy"]
```

Listing 9.17.Usuwanie elementów ze słownika

```
kraje["Cz"] =  nil
print(kraje)
//["PL": "Polska", "A": "Austria", "D": "Niemcy", "Sk": "Słowacja",
"Hr": "Chorwacja"]

//albo
if let usunietyElem = kraje.removeValue(forKey: "Cz") {
    print("Usunięto \(usunietyElem)")
} else {
    print("Brakuje elementu do usunicia")
}
```

Usunięcie elementu słownika polega na podstawieniu wartości *nil* dla konkretnego klucza. Można także zastosować metodę *updateValue(_:forKey:)*, która usuwa element, jeśli istnieje. Dlatego zastosowano instrukcję *if* do sprawdzenia, czy element istnieje (Listing 9.17).

Dostęp do kolejnych elementów słownika zapewniony jest poprzez instrukcję iteracyjną *for-in*. Na Listingu 9.18 przedstawiono kolejno wyświetlenie wszystkich elementów słownika (zarówno dla klucza i wartości), tylko wartości oraz tylko kluczy.

Listing 9.18.Dostęp do elementów ze słownika

```
for (symbol, panstwo) in kraje {
    print("\(symbol) - \(panstwo)")
}

for panstwo in kraje.keys {
    print("\(panstwo)")
}

for symbol in kraje.values {
    print(symbol)
}
```

Na podstawie słownika można utworzyć tablice zawierające albo klucze, albo wartości, co przedstawiono na Listingu 9.19.

Listing 9.19. Tworzenie tablicy ze słownika

```
let symbole = [String] (kraje.keys)
print(symbole)

let panstwa = [String] (kraje.values)
print(panstwa)
```

Zadanie 9.1.

Polecenie 1. Napisz program konsolowy, który zdefiniuje strukturę *liczbaZespolona* składającą się z części rzeczywistej i urojonej liczby.

Polecenie 2. Wczytaj dwie liczby zespolone.

Polecenie 3. Zaimplementuj dodawanie dwóch liczb zespolonych.

Polecenie 4. Zaimplementuj odejmowanie dwóch liczb zespolonych.

Polecenie 5. Zaimplementuj mnożenie dwóch liczb zespolonych.

Polecenie 6. Opracuj menu dla zaimplementowanych funkcjonalności.

Zadanie 9.2.

Polecenie 1. Napisz program konsolowy, który zdefiniuje strukturę *rzutOszczepem* składającą się z identyfikatora osoby oraz trzech prób rzutów.

Polecenie 2. Wczytaj dane dla czterech zawodników.

Polecenie 3. Wyświetl zawodnika, który uzyskał największą średnią swoich rzutów.

Polecenie 4. Usuń zawodnika, który uzyskał najmniejszą średnią swoich rzutów.

Zadanie 9.3.

Polecenie 1. Napisz program konsolowy, który zdefiniuje zbiór samochody składający się z modeli samochodów.

Polecenie 2. Wprowadź nazwy dla 6 samochodów.

Polecenie 3. Wyświetl wszystkie elementy zbioru.

Polecenie 4. Wczytaj model samochodu od użytkownika i usuń go ze zbioru. Należy sprawdzić, czy element taki istnieje w zbiorze.

Zadanie 9.4.

Polecenie 1. Napisz program konsolowy, który zdefiniuje słownik student, który zawiera nazwisko i identyfikator studenta.

Polecenie 2. Dodaj 5 studentów na podstawie wczytanych danych od użytkownika.

Polecenie 3. Wyświetl wszystkie dane.

Polecenie 4. Wczytaj dane studenta i wyszukaj go. Jeśli istnieje należy wyświetlić dane. W przeciwnym wypadku wyświetl odpowiednią informację.

Polecenie 5. Wczytaj dane studenta i jeśli istnieje, należy go usunąć.

Zadanie 9.5.

Polecenie 1. Napisz program konsolowy, który zdefiniuje strukturę *loty* składającą z miejsca wylotu (jako słownik: numer lotniska i jego nazwa), miejsca docelowego (jako słownik: numer lotniska i jego nazwa) oraz czas podróży w minutach.

Polecenie 2. Wczytaj dane dla *n* lotów do tablicy (*n* – podaje użytkownik). Należy sprawdzić poprawność danych.

Polecenie 3. Wyświetl dane wszystkich lotów.

Polecenie 4. Wyświetl dane wszystkich lotów, które trwają dłużej niż średnia wszystkich.

Polecenie 5. Usuń dane wszystkich lotów, które trwają krócej niż średnia wszystkich lotów.

Polecenie 6. Wyświetl dane lotów.