LABORATORIUM 10. DEFINIOWANIE KLAS – PROGRAMY ZORIENTOWANE OBIEKTOWO.

Cel laboratorium:

Poznanie funkcji i klas oraz posługiwanie się nimi w języku Swift.

Zakres tematyczny zajęć:

- tworzenie funkcji,
- tworzenie klas,
- inicjalizacja klas,
- dziedziczenie klas.

Pytania kontrolne:

1. Co to jest funkcja? W jakim celu się je tworzy?
1. Z czego składa się funkcja w języku Swift?
2. Jak utworzyć i wywołać funkcję bezparametrową?
3. Jak utworzyć i wywołać funkcję z parametrami?
4. Czy funkcja może zwracać wiele wartości?
5. Co to są klasy i jak je się stosuje? Co je odróżnia od struktur?
6. W jaki sposób są dziedziczone właściwości i funkcje?

Funkcja (ang. *Function*) to fragmenty kodu, które wykonują określone zadanie. Funkcja jest scharakteryzowana nazwą, która jest stosowana do wywołania funkcji w celu jej wykonania. Funkcja może być bezparametrowa lub posiadać parametry. Każda funkcja w języku Swift ma typ, składający się z typów parametrów funkcji i typu zwracanego przez funkcję. Można używać tego typu jak każdego innego typu w Swift, co ułatwia przekazywanie funkcji jako parametrów do innych funkcji i zwracanie funkcji z funkcji.

Przykład implementacji funkcji bezparametrowej oraz jej użycia w programie została przedstawiona na Listingu 10.1. Funkcja losuje liczbę całkowitą z zakresu 0-100. Jest ona wywołana wewnątrz instrukcji *print()*.

Listing 10.1. Funkcja bezparametrowa

```
func losujLiczbe() -> Int {
    return Int.random(in: 0..<101)
}

print("Wygenerowana liczba to \(losujLiczbe())")
```

Przykład funkcji z dwoma parametrami został przedstawiony na Listingu 10.2. Zwraca ona ciąg składający się z dwóch łańcuchów, podanych jako parametry funkcji. Należy pamiętać, aby przy wywołaniu podać nazwy argumentów.

Listing 10.2. Funkcja z 2 parametrami

```
func witaj(imie: String, nazwisko: String) -> String {
    let str = imie + " " + nazwisko + "- witaj na zajęciach z
programowania Swift!"
    return str
}
//wczytanie danych
print(witaj(imie: im!, nazwisko: nazw!))
```

Funkcja wyliczająca pole kwadratu na podstawie podanego boku została przedstawiona na Listingu 10.3. Wywołanie funkcji zostało podstawione do stałej.

Listing 10.3. Funkcja z parametrem typu całkowitego

```
func pole(bok: Int) -> Int {
    return bok * bok
}

let p = pole(bok: b!)

print("Pole kwadratu o boku \(b!) wynosi \(p)")
```

Funkcja nie musi zwracać wartości. Nie definiuje się typu zwracanego (po ->) i nie zwraca się wartości po słowie *return*. Na Listingu 10.4 przedstawiono wyświetlenie elementów tablicy.

Listing 10.4. Funkcja, która nie zwraca wartości

```
func wyswietl(tab: [Int]) {

    print("Elementy tablicy:")
    for i in tab {
      print(i)
    }
}

wyswietl(tab: num)
```

Funkcja może zwracać więcej niż jedną wartość, korzystając z typu krotki. Na Listingu 10.5 przedstawiono funkcję, która oblicza sumę elementów parzystych oraz sumę elementów nieparzystych elementów tablicy.

Listing 10.5. Funkcja zwracająca wiele wartości

```
func suma(tab: [Int]) -> (Int, Int) {
    var sump = 0
    var sumnp = 0

    for i in tab {
        if i % 2 == 0 {
            sump += i
        } else {
            sumnp += i
        }
    }
    return (sump, sumnp)
}

let num = [1, 6, 8, 9, 33, 11, 70]
var sum: (Int, Int) = suma(tab: num)
print("Suma elementów parzystych: \(sum.0), suma elementów
nieparzystych: \(sum.1)")
```

Może się zdarzyć, że krotka nie może zostać zwrócona przez funkcję. Jeśli funkcja ma obliczyć średnią elementów dodatnich oraz średnią elementów ujemnych, a tablica wejściowa będzie zawierała same zera, funkcja nie może zwrócić wartości. Należy wtedy zastosować opcjonalność dla zwracanej krotki, co zostało przedstawione na Listingu 10.6. Przy wywołaniu funkcji zastosowano instrukcję *if* do sprawdzenia, czy dane istnieją.

Listing 10.6. Funkcja zwracająca wiele wartości

```
func srednia(tab: [Int]) -> (srdod: Double, srujem:Double)? {
    if tab.isEmpty {
        return nil
    }
    var sumd = 0, sumu = 0, ld = 0, lu = 0
    for i in tab {
        if i > 0 {
            sumd += i
            ld += 1

        } else {
            sumu += i
            lu += 1
        }
    }
    if (ld != 0){
        if (lu != 0){
```

```
            return (Double(sumd/ld), Double(sumu/lu))
        }else {
            return(nil)
        }
    } else {
        return (nil)
    }
}
let num = [0, 0, 0, 0]
if let sr = srednia(tab: num) {
    print("Średnia elementów dodatnich: \(sr.srdod), a ujemnych:
\(sr.srujem)")
} else {
    print("Nie można obliczyć obu średnich")
}
```

Parametry funkcji mogą posiadać wartości domyśle, co przedstawiono na Listingu 10.7. Przy wywołaniu można pominąć te argumenty, dla których nie zostaną podane wartości. Zostaną wtedy uwzględnione wartości podane dla parametrów.

Listing 10.7. Funkcja z domyślnymi wartościami parametrów

```
func suma(a: Int = 0, b: Int = 0) -> Int {
    return a + b
}

var x: Int, y: Int
print(suma())
```

Domyślnie wartości parametrów nie mogą być modyfikowane wewnątrz funkcji i zwracane do programu głównego, skąd następuje wywołanie funkcji. Dodając słowo kluczowe *inout* do parametru funkcji pozwala na zmianę tych zmiennych. Na Listingu 10.8 przedstawiono funkcję, która zamienia wartościami 2 zmienne. Ponieważ są one typu *in-out*, wynik zmiany zostanie przekazany do głównego programu. Należy pamiętać, aby przy wywołaniu zastosować *&*.

Listing 10.8. Funkcja z parametrami typu in-out

```
func zamiana(x: inout Int, y: inout Int) {
    let tmp = x
    x = y
    y = tmp
}

var a = 18, b = 90
print("a = \(a), b = \(b)") // a = 18, b = 90
zamiana(x: &a, y: &b)
print("a = \(a), b = \(b)") // a = 90, b = 18
```

Klasy (ang. *Classes*) to konstrukcje, które pozwalają na definiowanie właściwości, metod, ale także umożliwiają dziedziczenie czy rzutowanie typów.

Przykład klasy *Samochod* składający się z 5 właściwości został przedstawiony na Listingu 10.9. Należy pamiętać o przypisaniu wartości domyślnych. Deklaracja instancji klasy jest analogiczna do deklaracji struktury.

Listing 10.9. Klasa Samochod

```
class Samochod {
    var marka: String = ""
    var model: String = ""
    var rokProd: Int = 0
    var przebieg: Double = 0.0
    var cena: Double = 0.0
}
var s = Samochod()
```

Przypisanie wartości do instancji klasy następuje poprzez operator kropki (Listing 10.10).

Listing 10.10. Przypisanie wartości

```
 s.marka = "Porsche"
 s.model = "911"
 s.rokProd = 2000
 s.przebieg = 10000
 s.cena = 120000
```

Klasa, jak struktura, ma domyślną inicjalizację danych. Przykład tworzenia metody *init()* został przedstawiony na Listingu 10.11. do wskazania właściwości należy zastosować *self*, który wskazuje na właściwość klasy.

Listing 10.11. Inicjalizacja

```
class Samochod {
     …
     init(marka: String, model: String, rokProd: Int, przebieg:
          Double, cena: Double) {
         self.marka = marka
         self.model = model
         self.rokProd = rokProd
         self.przebieg = przebieg
         self.cena = cena
     }
}
```

```
var s = Samochod(marka:"Porsche", model: "911", rokProd: 2000,
przebieg: 10000.0, cena: 12000.0)
```

Na Listingu 10.12 przedstawiono funkcję, która wyświetla wszystkie elementy klasy. Jej wywołanie zachodzi poprzez instancję klasy.

Listing 10.12. Definiowanie funkcji klasy

```
class Samochod {
    …

    func wyswietl() {
        print("\(marka) \(model)")
        print("Rok produkcji: \(rokProd)")
        print("Przebieg: \(przebieg)")
        print("Cena: \(cena)")
    }
}

var s = Samochod(marka:"Porsche", model: "911", rokProd: 2000,
przebieg: 10000.0, cena: 12000.0)

s.wyswietl()
```

Listing 10.13. Dziedziczenie

```
class Kwadrat {
    var a: Double = 0.0
    init(bok: Double) {
        self.a = bok
    }

    func pole() -> Double {
        return a * a
    }
}

class Szescian : Kwadrat {
    override func pole() -> Double {
        return 6 * super.pole() //return 6 * a * a
    }

    func objetosc() -> Double {
        return a * a * a
    }
}
```

```
var kw = Kwadrat(bok: 2.5)
print(kw.pole())
var sz = Szescian(bok: 5.0)
print(sz.pole())
print(sz.objetosc())
```

Ważnym elementem klas jest dziedziczenie właściwości oraz funkcji jednej klasy przez inne. Przykład dziedziczenia został przedstawiony na Listingu 10.13. Klasa *Kwadrat* jest scharakteryzowany poprzez jego bok (*a*). W klasie zdefiniowano metodę inicjującą oraz funkcję obliczającą pole. Druga klasa *Szescian* dziedziczy po klasie *Kwadrat* (*Szescian: Kwadrat*). Nie zdefiniowano w niej żadnych właściwości, bo korzysta ona z tych zdefiniowanych w klasie *Kwadrat*. Klasa *Szescian* zawiera dwie funkcje: *objetosc()*, zdefiniowana na potrzeby tej klasy oraz *pole()*, która modyfikuje funkcję dziedziczoną. Funkcja *pole()* ma taką samą nazwę, jak w klasie *Kwadrat*. Ponieważ zwracana jest inna wartość, funkcję należy nadpisać, stosując słowo kluczowe *override*. Implementację tej funkcji można wykonać korzystając z już utworzonej funkcji *pole()* klasy *Kwadrat*. Aby ją wywołać należy wskazać, że pochodzi z klasy, po której się dziedziczy poprzez słowo kluczowe *super*. Przykłady użycia klas zostały także przedstawiona na Listingu 10.13.

Zadanie 10.1.

Polecenie 1. Napisz program konsolowy, który wygeneruje 3 liczby losowe z zakresu 1-250. Napisz funkcję, która zwróci trzy liczby całkowite.

Polecenie 2. Napisz funkcję, którą zwróci największą liczbę z trzech podanych. Użyj utworzonej funkcji do wyświetlenia wyniku.

Polecenie 3. Napisz funkcję, którą zwróci najmniejszą liczbę z trzech podanych. Użyj utworzonej funkcji do wyświetlenia wyniku.

Polecenie 4. Napisz funkcję, którą zwróci jednocześnie najmniejszą oraz największą liczbę z trzech podanych. Użyj utworzonej funkcji do wyświetlenia wyniku.

Zadanie 10.2.

Polecenie 1. Napisz program konsolowy, który wyświetli menu w postaci:

1 – dodanie

2 – odejmowanie

3 – mnożenie

4 – dzielenie

5 – pierwiastkowanie

Polecenie 2. Dla każdej ewentualności należy wczytać dwie liczby zmiennoprzecinkowe lub jedną (dla pierwiastkowania). Należy napisać odpowiednie funkcje.

Zadanie 10.3.

Polecenie 1. Napisz program konsolowy, który wczyta współrzędne dwóch punktów układu współrzędnych. Należy napisać funkcję wczytującą współrzędne i je zwracające.

Polecenie 2. Napisz funkcję obliczającą odległość między wczytanymi punktami.

Polecenie 3. Napisz funkcję zwracającą numer ćwiartki, w której leży dany punkt.

Polecenie 4. Napisz funkcję, która sprawdzi, czy dany punkt leży wewnątrz kwadratu, na jego obwodzie, czy poza nim. Należy podać jedną współrzędną wierzchołka kwadratu. Kwadrat jest symetryczny względem początku układu współrzędnych.

Polecenie 5. Przetestuj utworzone funkcje.

Zadanie 10.4.

Polecenie 1. Napisz program konsolowy, który wczyta liczbę elementów tablicy oraz jej elementy całkowite, a następnie wyznaczy największy element, najmniejszy element oraz średnią arytmetyczną wszystkich elementów.

Polecenie 2. Napisz funkcję, która wczyta elementy do tablicy i zwróci tą tablicę. Funkcja powinna posiadać parametr mówiący o liczbie elementów tablicy.

Polecenie 3. Napisz funkcję, która wyświetli wszystkie elementy tablicy.

Polecenie 4. Napisz funkcję, która wyznaczy i zwróci najmniejszy element tablicy oraz indeks, na którym się znajduje.

Polecenie 5. Napisz funkcję, która wyznaczy i zwróci największy element tablicy oraz indeks, na którym się znajduje.

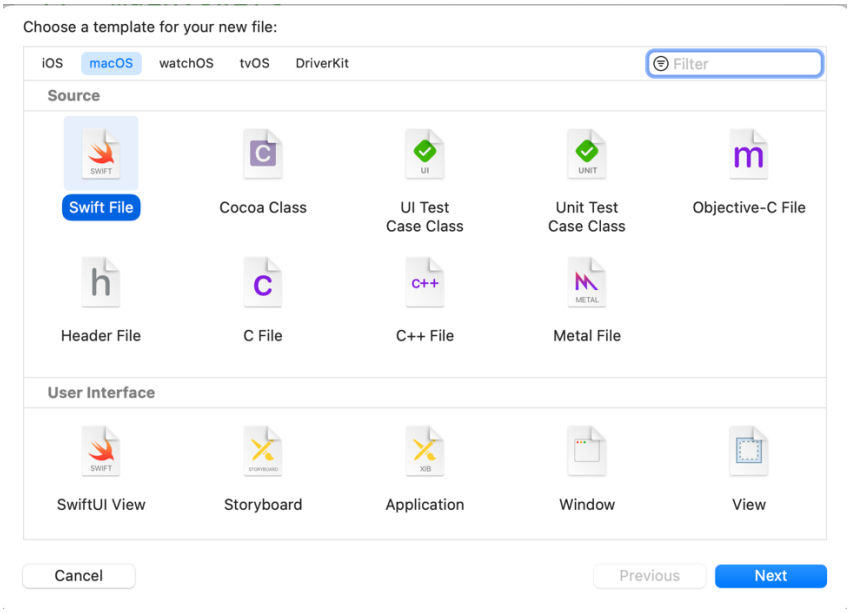
Polecenie 6. Napisz funkcję, która zamieni miejscami element największy z najmniejszym.

Polecenie 7. Napisz funkcję, która wyznaczy i zwróci średnią arytmetyczną wszystkich elementów tablicy.

Polecenie 8. Przetestuj utworzone funkcje.

Zadanie 10.5.

Polecenie 1. Utwórz klasę *Osoba*, zawierającą imię, nazwisko, rok urodzenia. Dodawanie klasy zostało przedstawione na rysunku 10.1.



Rys. 10.1. Dodanie nowej klasy

Polecenie 2. Utwórz metodę *init()*.

Polecenie 3. Utwórz funkcję obliczającą wiek osoby.

Polecenie 4. Utwórz funkcję wyświetlającą dane osoby.

Polecenie 5. Przetestuj utworzone funkcje. Zdefiniuj dwie osoby. Wyświetl, która osoba jest starsza, a która młodsza, lub obie, jeśli są w tym samym wieku.

Zadanie 10.6.

Polecenie 1. Utwórz klasę *Pracownik*, dziedziczącą po klasie *Osoba* i zawierającą rok zatrudnienia, stanowisko, stawkę za godzinę i liczbę zrealizowanych godzin oraz nazwę firmy. Stanowisko powinno być z góry zdefiniowane jako typ wyliczeniowy.

Polecenie 2. Utwórz metodę *init()* korzystając z tej z klasy *Osoba*.

Polecenie 3. Utwórz funkcję zwracającą liczbę lat pracy w danej firmie.

Polecenie 4. Utwórz funkcję zwracającą wysokość pensji za przeprowadzone godziny, według stawki.

Polecenie 5. Utwórz funkcję wyświetlającą dane pracownika, nadpisując funkcję z klasy Osoba.

Polecenie 6. Przetestuj utworzone funkcje.

LABORATORIUM 11. DEFINIOWANIE KLAS – PROGRAMY ZORIENTOWANE OBIEKTOWO.

Zadanie 11.1.

Polecenie 1. Utwórz klasę *Prostokąt*, scharakteryzowaną przez długości boków.

Polecenie 2. Utwórz metodę *init()*.

Polecenie 3. Utwórz funkcję zwracającą pole figury.

Polecenie 4. Utwórz funkcję zwracającą obwód figury.

Polecenie 3. Utwórz funkcję wyświetlająca dane figury.

Polecenie 7. Przetestuj utworzone funkcje.

Zadanie 11.2.

Polecenie 1. Utwórz klasę *Prostopadłościan*, dziedziczącą po klasie *Prostokat* i scharakteryzowaną dodatkowo przez wysokość.

Polecenie 2. Utwórz metodę *init()*.

Polecenie 3. Utwórz funkcję zwracającą pole całkowite bryły.

Polecenie 4. Utwórz funkcję zwracającą sumę krawędzi.

Polecenie 5. Utwórz funkcję zwracającą objętość bryły.

Polecenie 6. Utwórz funkcję wyświetlająca dane bryły.

Polecenie 7. Przetestuj utworzone funkcje.

Zadanie 11.3.

Polecenie 1. Utwórz klasę *Student*, dziedziczącą po klasie *Osoba* (zad. 10.5) i scharakteryzowaną przez: numer indeksu, kierunek studiów, rok studiów, oceny z 5 przedmiotów.

Polecenie 2. Utwórz metodę *init()*. Kierunek studiów powinien być zdefiniowany jako typ wyliczeniowy. Należy zapewnić poprawność danych wprowadzonych ocen.

Polecenie 3. Utwórz funkcję wyznaczającą średnią studenta na podstawie jego ocen.

Polecenie 4. Utwórz funkcję wyświetlającą dane studenta.

Polecenie 5. Utwórz tablicę studentów. Liczbę studentów należy pobrać od użytkownika. Należy wczytać dane studentów. Należy utworzyć odpowiednią funkcję.

Polecenie 6. Wyświetl wszystkich studentów studiujących na podanym przez użytkownika kierunku. Należy utworzyć odpowiednią funkcję.

Zadanie 11.4.

Polecenie 1. Utwórz klasę *StudentNaErasmusie*, dziedziczącą po klasie *Student* (zad. 11.3) i scharakteryzowaną przez: nazwę uczelni za granicą, datę rozpoczęcia Erasmusa, datę zakończenia Erasmusa oraz kursy w których student brał udział (w postaci tablicy zawierającej nazwę kursu oraz uzyskanej oceny).

Polecenie 2. Utwórz metodę *init()*.

Polecenie 3. Utwórz funkcję wyświetlającą dane studenta na Erasmus, korzystając z funkcji dziedziczonych.

Polecenie 4. Utwórz funkcję wyznaczającą czas spędzony na Erasmusie.

Polecenie 5. Utwórz funkcję wyznaczającą ocenę studenta na podstawie uczestnictwa w kursach.

Średnia	**Ocena**
[4,6-5,0]	bardzo dobra
[3,6-4,5]	dobra
[3,0-3,5]	dostateczna
< 3,0	niedostateczna