



Programowanie mobilne

Wprowadzenie do języka Swift

dr inż. Andrzej Wilczyński - www.awilczynski.me

Swift

Język programowania urządzeń dla systemów macOS, iOS, watchOS, tvOS stworzony przez firmę Apple i zaproponowany po raz pierwszy w roku 2014. Został zaprojektowany z uwzględnieniem filozofii łatwości użycia i szybkości działania języków skryptowych, a jednocześnie oferuje potężne możliwości charakterystyczne dla języków kompilowanych. Kompilator został zoptymalizowany pod kątem wydajności, a język zoptymalizowany pod kątem programowania, bez kompromisów w obu przypadkach. Najpopularniejszym środowiskiem wykorzystywanym do programowania w Swiftcie jest Xcode.

Dokumentacja dostępna na:

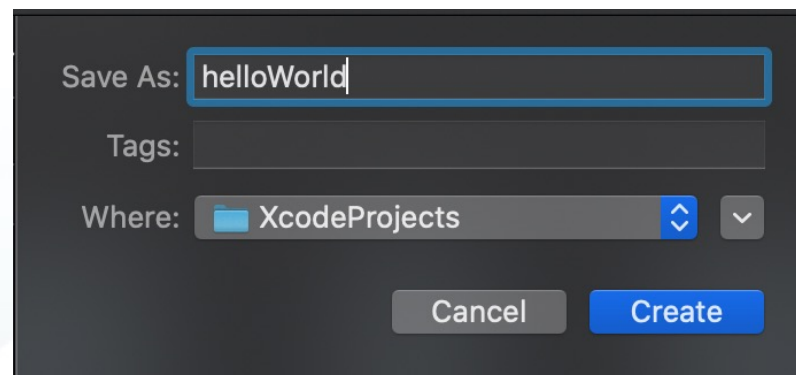
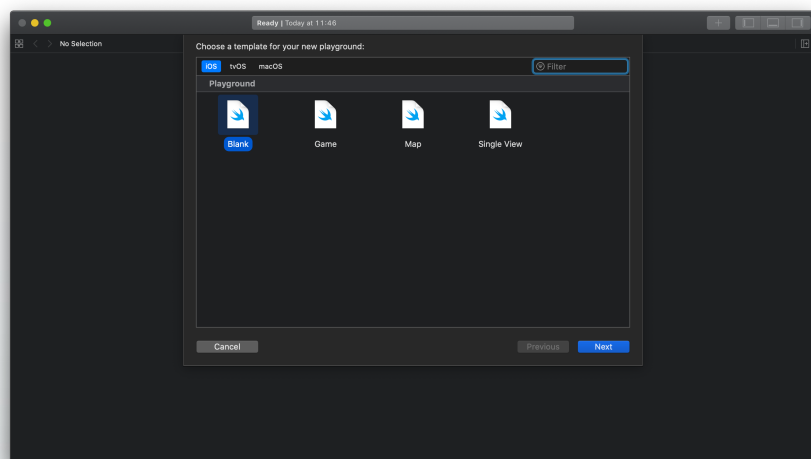
<https://docs.swift.org/swift-book>

Pierwszy plik źródłowy w Swift (1)

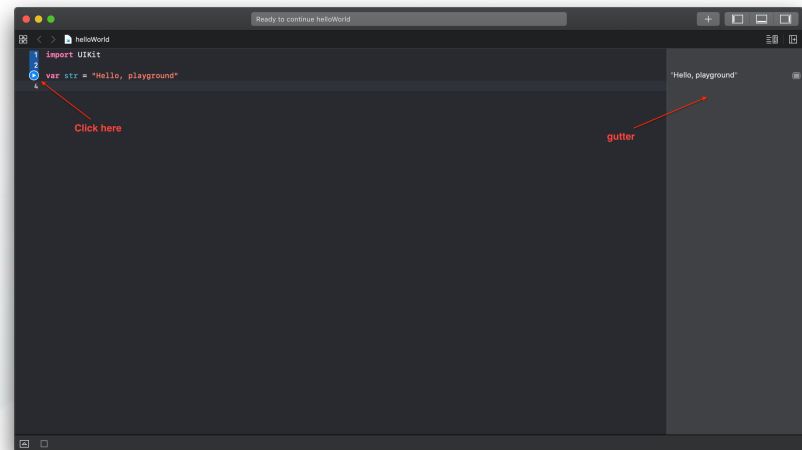
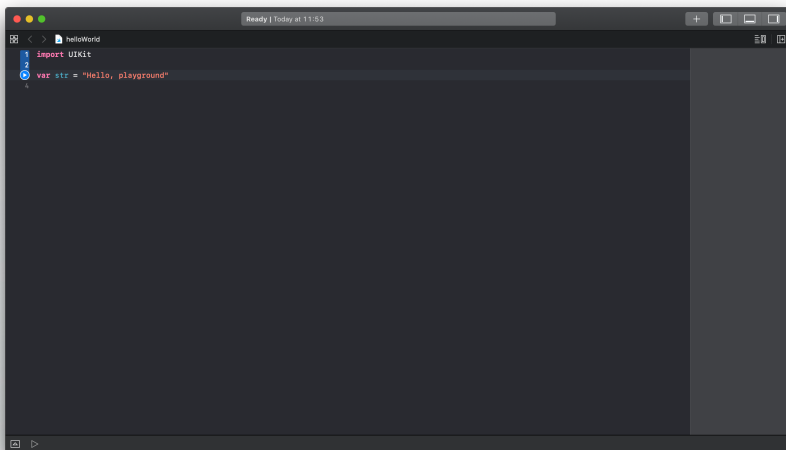
Tworzymy projekt za
pomocą przycisku
„Create New Project”.



Pierwszy plik źródłowy w Swift (2)



Pierwszy plik źródłowy w Swift (3)



Deklarowanie zmiennych

Zmienne używane są do przechowywania wartości, wykorzystywanych przez program na dalszym etapie.

Przykład:

```
var str = "Hello, playground"  
var sum = 5 + 7
```

Główne typy zmiennych języka Swift

- Int – liczby całkowite – 12, -25
- Float, Double – liczby dziesiętne – 3.14, 2.56
- Boolean – wartości logiczne – true, false
- Character – pojedynczy znak - 'x'
- String – kolekcja znaków - "mobilki", "komorka"
- Let – stała - 3.1415

Wyraźne określanie typu zmiennej

W takim przypadku typ zmiennej jest podawany w jej deklaracji.

Przykład:

```
var studentAge = 40  
var numOfPoints:Int = 77
```


Interpolacja ciągu tekstowego

Dzięki interpolacji w ciągu tekstowym można bardzo łatwo używać stałych, zmiennych, literałów i wyrażeń jako literałów.

Przykład:

```
var cookies = 5  
print("Student has " +  
String(cookies) + " cookies.")
```

Przykład:

```
var cookies = 5  
print("Student has \$(cookies) cookies.")
```

Polecenie `if`

```
var points = 5
if points > 2
{
    print("Student passed the course")
} else
{
    print("Student did not pass the course")
}
```

Konstrukcja switch

```
var grade = "A"
switch grade
{
    case "A":
        print("Excellent achievements.")
    case "B":
        print("Better than average.")
    case "C":
        print("Average score.")
    case "D":
        print("You still have to work on yourself.")
    case "F":
        print("Failure, you must take the course again.")
    default:
        print("Incorrect rating provided.")
}
```

Pętla while

```
var testScores = [85,70,92,100]
var i = 0
while i < 4
{
    print(testScores[i])
    i += 1
}
```

Pętla for

```
var testScores = [85,70,92,100]
for score in testScores {
    print(score)
}
```


Tablice

Tablica jest nazywana i deklarowana w sposób podobny do zmiennej lub stałej, ale stanowi zupełnie inny typ danych. W języku Swift tablica zalicza się do typu kolekcji.

Przykład:

```
var students:[String] = ["Andrew", "Hose", "Alice"]  
var points:[Float] = []  
points.append(4.5)
```

Słowniki

Słownik jest podobny do tablicy, z tym, że wartości są przechowywane w postaci par klucz-wartość opartych na unikatowych kluczach.

Przykład:

```
var students:[String : Float] = ["Andrew":5.00,"Hose":3.00,"Alice":3.5]
for (name, points) in students
{
    print("\(name) has \(points) points.")
}
```

Funkcje

Funkcja to nazwany i hermetyzowany blok kodu przeznaczony do wielokrotnego użycia. Funkcja może pobierać parametry i zwracać wartości, ale nie musi.

Przykład:

```
func sayHello()  
{  
    print("Hello!")  
}
```

```
sayHello()
```

Funkcje zwracające wartość

W przypadkach, kiedy konieczne jest wykorzystanie wyniku działania funkcji, musimy utworzyć funkcję oferującą wartość zwrótną.

Przykład:

```
func sayGreeting(name: String)->String
{
    return "Hello, \(name)"
}
```

```
print(sayGreeting(name: "Andrew"))
```

Funkcje i zasięg zmiennej lub stałej

```
var score = 1000  
let width = 10.55
```

```
func myFunc()  
{  
    var funcVar = 1000  
    print("Variable outside the function: \(score)")  
    print("Constant outside the function: \(width)")  
    print("FuncVar variable inside a function: \(funcVar)")  
}
```

```
print("Variable outside the function: \(score)")  
print("Constant outside the function: \(width)")  
print("FuncVar variable inside a function: \(funcVar)")  
-> błąd "Use of unresolved identifier 'funcVar'"
```


Funkcje zagnieżdżone

Funkcja zagnieżdżona to funkcja zadeklarowana wewnątrz bloku kodu innej funkcji.

Przykład:

```
func nested(i:Int, j:Int)
{
    func printAnswer(answer: Int)
    {
        print("\(answer)")
    }
    printAnswer(answer: i + j)
}
nested(i: 6, j: 7)
```

Typ wyliczeniowy

Typ wyliczeniowy to zdefiniowany ogólny typ danych przeznaczony dla grupy powiązanych ze sobą wartości, których można używać w sposób zapewniający bezpieczeństwo typu.

Przykład:

```
enum Day
{
    case Monday
    case Tuesday
    case Wednesday
    case Thursday
    case Friday
    case Saturday
    case Sunday
}
print (Day.Monday);
```

Klasy

Klasa to moduł kodu zaprojektowany w celu przedstawienia w aplikacji rzeczywistego obiektu. Klasy są uznawane za elementy konstrukcyjne aplikacji.

Przykład:

```
class Student
{
    var name:String = "Andrew"
    var lastname:String = "Kowalski"
    var points:Int = 2
}
```

Klasy – metoda inicjalizacyjna

Metoda inicjalizacyjna to specjalny typ metody klasy zdefiniowanej za pomocą słowa kluczowego *init* i wykonywanej w trakcie tworzenia każdego egzemplarza danej klasy.

Przykład:

```
class Student
{
    var name:String
    var Lastname:String
    var points:Int = 2

    init(name:String, lastname:String)
    {
        self.name = name
        Lastname = lastname
    }
}
```

Klasy – metody

```
class Student
{
    var name:String
    var Lastname:String
    var points:Int = 2

    init(name:String, lastname:String)
    {
        self.name = name
        Lastname = lastname
    }

    func getName()->String
    {
        return name
    }

    func learn()
    {
        print("\(name) \((Lastname) is learning.")
    }
}
```


Klasy – tworzenie obiektu

W celu użycia klasy konieczne jest utworzenie jej egzemplarza.

Przykład:

```
let student = Student(name:"Andrew", lastname:"Kowalski")  
print(student.getName())  
student.learn()
```

Dziedziczenie (1)

Podklasy to hierarchiczne klasy potomne klasy nadrzędnej, współdzielą kluczowe właściwości i metody z klasą nadrzędną, ale jednocześnie na tyle się od niej odróżniają, że uzasadnione jest utworzenie oddzielnej klasy, jest to mechanizm określany jako dziedziczenie.

Dziedziczenie (2)

```
class Person
{
    var name:String
    var lastname:String

    init(name:String, lastname:String)
    {
        self.name = name
        self.lastname = lastname
    }
}
```

```
class Student:Person
{
    var points:Int

    init(name:String, lastname:String,
        points:Int)
    {
        self.points = points
        super.init(name: name, lastname:
            lastname)
    }

    func result()
    {
        print("\(name) \(lastname) has \(points)
            points.")
    }
}
```

```
let student = Student(name: "Andrew", lastname: "Kowalski", points: 5)
student.result()
```

Protokoły (1)

Protokół w języku Swift to element programu definiujący, że określone właściwości i metody muszą znajdować się w zgodnych z nim klasach.

Przykład:

```
protocol Person
{
    var name: String { get }
    var lastname: String { get }

    func result()
}
```

Protokoły (2)

```
protocol Person
{
    var name: String { get }
    var lastname: String { get }

    func result()
}
```

```
class Student:Person
{
    var name:String
    var lastname:String
    var points:Int?

    init(name: String, lastname: String, points:
        Int? = nil)
    {
        self.name = name;
        self.lastname = lastname
        self.points = points
    }

    func result()
    {
        print("\(name) \(lastname) has \(points)
            points.")
    }
}
```

```
let student = Student(name: "Andrew", lastname: "Kowalski", 5)
student.result()
```


Struktury

Struktury w języku Swift są przeznaczone do hermetyzacji niewielkich zbiorów prostych wartości. Struktury można porównać do klas, ponieważ mogą zawierać właściwości i metody.

Przykład:

```
struct Vector  
{  
    var x = 0  
    var y = 0  
    var z = 0  
}
```

```
var v1 = Vector(x:5, y:8, z:5)
```

Przeciążanie operatora (1)

Technika dodawania funkcjonalności do operatorów i zwiększania ich możliwości zwłaszcza podczas pracy z wygenerowanymi przez użytkownika typami wyliczeniowymi, klasami i strukturami.

Przykład:

```
func + (augend: Vector, addend: Vector) -> Vector
{
    return Vector(x: augend.x + addend.x, y: augend.y + addend.y, z:
        augend.z + addend.z)
}
```

Przeciążanie operatora (2)

```
struct Vector
```

```
{  
    var x = 0  
    var y = 0  
    var z = 0  
  
    func test()  
    {  
        print(x)  
    }  
}
```

```
func + (augend: Vector, addend: Vector) -> Vector
```

```
{  
    return Vector(x: augend.x + addend.x, y: augend.y + addend.y, z: augend.z + addend.z)  
}
```

```
var v1 = Vector(x:5, y:8, z:5)  
var v2 = Vector(x:9, y:4, z:2)  
print(v1+v2)
```

Funkcje generyczne

Funkcje generyczne w języku Swift to funkcje, które mogą działać ze wszystkimi zgodnymi z nimi typami danych.

Przykład:

```
func isEqual<T: Equatable>(a:T, b:T) -> Bool
{
    return a == b
}
```

```
print(isEqual(a: 3, b: 3))
print(isEqual(a: 3.4, b: 3.8))
```

Kontrola dostępu

- Open
- Public
- Internal (domyślny)
- File-private
- Private

Kontrola dostępu - Open

- Klasy z tym modyfikatorem mogą być dziedziczone wszędzie
- Właściwości i metody z tym modyfikatorem mogą być przesłonięte wszędzie

Przykład:

```
open class Person{}
```

```
open func result() {}
```



Kontrola dostępu - Public

- Klasy z tym modyfikatorem mogą być dziedziczone tylko w swoim module
- Właściwości i metody z tym modyfikatorem mogą być przesłonięte tylko w swoim module

Przykład:

```
public class Person{}  
  
public func result(){}  

```

Kontrola dostępu – internal (domyślny)

- Klasy z tym modyfikatorem mogą być używane tylko w swoim module
- Właściwości i metody z tym modyfikatorem mogą być używane tylko w swoim module

Przykład:

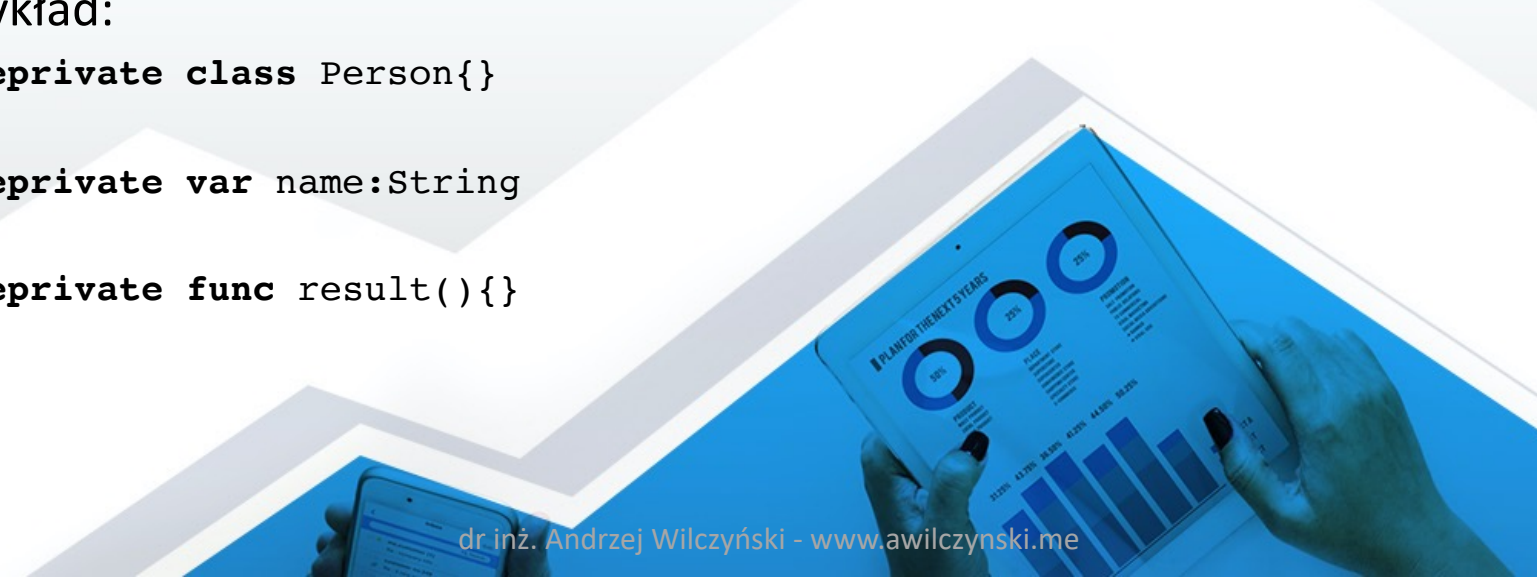
```
internal class Person{}  
  
internal func result(){}  

```

Kontrola dostępu – fileprivate

- Klasy z tym modyfikatorem mogą być używane tylko w pliku w którym są zdefiniowane
- Właściwości i metody z tym modyfikatorem mogą być używane tylko w pliku w którym są zdefiniowane

Przykład:

```
fileprivate class Person{}  
  
fileprivate var name:String  
  
fileprivate func result(){}  
  

```

Kontrola dostępu – private

- Właściwości i metody z tym modyfikatorem mogą być używane tylko wewnątrz konkretnej klasy

Przykład:

```
private var name:String
```

```
private func result(){}
```

Dziękuję! 😊

Bibliografia

1. Mark A. Lassoﬀ, Tom Stachowitz, *Podstawy języka Swift. Programowanie aplikacji dla platformy iOS*, Helion, 2016