

*PG5600* **ios**

**programming**

**Forelesning 2**

# Forrige gang

- økosystemet rundt iOS-utvikling
- Xcode og Playgrounds
- Swift
  - Stringer
  - loops
  - if & switch
  - Optionals
  - \* **Numbers**

# Funksjoner

```
// Globale som standard
```

```
func hello() {  
    print("Hello")  
}
```

```
hello()
```

# Funksjoner med returverdi

```
func hello() -> String {  
    return "Hello World"  
}
```

```
print(hello())
```

# Funksjoner med optional returverdi

```
func hello() -> String? {  
    return nil  
}
```

```
print(hello())
```

# Funksjoner med multiple return-verdier

```
func error() -> (code: Int, message: String) {  
    return (500, "Internal server error")  
}
```

```
print(error().message)
```

# Funksjoner med parametere

```
func greet(prefix: String, name: String) {  
    print("Hello, \(prefix) \(name)!")  
}
```

```
greet(prefix: "Mr", name: "Anderson")
```



# Funksjoner med optional parameter

```
func greet(prefix: String?, name: String) {  
    if let actualPrefix = prefix {  
        print("Hello, \(actualPrefix) \(name)!")  
    } else {  
        print("Hello \(name)!")  
    }  
}
```

```
greet(prefix: "Mr", "Anderson")  
greet(prefix: nil, "Anderson")
```

# Funksjoner med default-parametere

Defaultparamtere til slutt:

```
func greet(name: String, prefix: String = "") {  
    print("Hello, \(prefix) \(name)!")  
}
```

```
greet(name: "Anderson")
```

```
greet(name: "Anderson", prefix: "Mr")
```

# Utelat navn ved kall

Som regel ikke å anbefale pga. lesbarhet, men greit å kjenne til

// Merk \_ foran prefix:

```
func greet(name: String, _ prefix: String = "") {  
    print("Hello, \(prefix) \(name)!")  
}
```

```
greet(name: "Anderson")
```

```
greet(name: "Anderson", "Mr")
```

# rename parameterere

```
func greet(prefix p: String, name n: String) {  
    print("Hello, \(p) \(n)!")  
}
```

```
// Navn MÅ brukes når funksjonen kalles  
greet(prefix: "Mr", name: "Anderson")
```

# **Playground demo**

## **funksjoner**

# Funksjoner med varargs (variadic)

```
func greet(names: String...) {  
    for name in names {  
        print("Hello \ (name)")  
    }  
}
```

```
greet(names: "Agent Smith", "Mr. Anderson")
```

- Maks ett
- Alltid til slutt i parameterlisten (også etter default parameterer, om de finnes)

# Funksjoner som endrer på parameterne, eksternt

```
func swapInts(first: inout Int, second: inout Int) {  
    let temp = first  
    first = second  
    second = temp  
}
```

```
var a = 10  
var b = 5  
// Må kalles med & foran parametere  
swapInts(first: &a, second:&b))  
// a = 5  
// b = 10
```

**Funksjoner som  
returnerer/tar imot  
andre funksjoner**



```
func createFunction() -> () -> String {
    func helloWorld() -> String {
        return "Hello world"
    }
    return helloWorld
}

func invokeFunction(fn: () -> String, times: Int) {
    for _ in 0 ..< times {
        print(fn())
    }
}

// Funksjon som lager en funksjon
let fn = createFunction()

// Kan kalles direkte
print(fn())

// Eller sendes videre til annen funksjon
invokeFunction(fn: fn, times: 3)
```

# Closures

**Aka blokker (obj-c), lamdas,  
anonyme funksjoner**

# Closures - syntaks

```
{ (parameter) -> returntype in  
    uttrykk  
}
```

```
let greetingClosure = { (greeting : String) -> Void in  
    print(greeting)  
}
```

```
greetingClosure("Hei")
```

# Closures - syntaks

// På Array:

```
public func sorted(by: (Int, Int) -> Bool) -> [Int]
```

```
var numbers = [43,2,1,90]
```

```
numbers.sorted(by { x, y in
```

```
    if y > x {
```

```
        return true
```

```
    } else {
```

```
        return false
```

```
    }
```

```
})
```

```
// 1,2,43,90
```

# Closures - syntaks

```
let arr = [43, 2, 1, 90]
```

```
arr.sorted(by: { x, y in y > x }) // 1, 2, 43, 90
```

- Type inference
- Implisitt return for enkeltlinje-uttrykk

# Closures - syntaks

```
let numbers = [1, 2, 3, 4, 5]
```

```
// -> [10, 20, 30, 40, 50]
```

```
numbers.sort{ $0 < $1 }
```

```
// kan droppe x, y hvis closure er siste argument)
```

- Shorcuts til parameternavn \$0, \$1, \$2, osv.

# Enumerations

```
enum Vaskeprogram {  
    case ikkeValgt  
    case bomull  
    case ull  
    case syntetisk  
}
```

```
var program = Vaskeprogram.ikkeValgt  
program = .bomull
```



# Classes and structs

```
struct Coordinate {  
    //... ?  
}
```

```
class Person {  
    //... ?  
}
```

	Klasser	Structs
Properties	✓	✓
Metoder	✓	✓
Subscripts	✓	✓
Initializers	✓	✓
Extendable	✓	✓
Støtte for protokoller	✓	✓
Arv	✓	
Type casting	✓	
Deinitializers	✓	
Flere referanser til samme instans	✓	

# Bruk structs når

- Hovedhensikten er å strukturere noen enkle verdier
- Du ønsker at dataene skal kopieres fremfor refereres
- Du ikke trenger arv

I andre tilfeller: bruk klasser

# Structs

```
struct PointOfInterest {  
    var latitude: Double = 0  
    var longitude: Double = 0  
    var name : String  
}
```

// har initializer som standard:

```
let poi1 = PointOfInterest(latitude: 59.91126, longitude: 10.76046, name: "Westerdals")  
print("\(poi1.name) - \(poi1.latitude), \(poi1.longitude)")
```

```
var poi2 = poi1 // <-- kopi
```

```
poi2.name = "NITH"
```

```
// p1.name er fortsatt Westerdals
```

```
// p2.name er ?
```

# Pass by value vs. reference

- Structs (inkl. Stringer, Arrays og Dictionaries), Int og Enums er datatyper med pass by value, og som kopieres når de sendes rundt
- Ikke så skummelt som det høres ut som, Swift optimaliserer slik at kopiering bare skjer når det er helt nødvendig
- Klasser, funksjoner og closures sendes som referanser og kopieres ikke

# Klasser

```
// Deklarere
class Server {
    // Stored properties - ikke instansvariabler
    var ip: String
    var startTime : Date?
    var running = false

    // Konstruktør
    init(ip: String) {
        self.ip = ip
    }
}

// Instansiere
let server = Server(ip: "192.168.0.1")
```

# Metoder

```
class Server {  
    // ...  
    func boot() {  
        startTime = Date()  
    }  
}
```

```
let server = Server(ip: "192.168.0.1")  
server.boot()
```



# Metoder

- For structs, klasser, enums
- Samme syntaks som funksjoner
- `self` refererer til instansen, på samme måte som `this` i andre språk
- Eksplisitt navngiving som standard, for 2...N parameter:

```
func install(package: String, version: String) {}  
server.install("swift", version: "1.0.0")
```

# Properties

- Stored properties (klasser, structs)
- Computed properties (klasser, structs og enums)

# Computed properties

```
class Server {  
    // ...  
    // computed properties  
    var uptime : Int {  
        get {  
            if let start = startTime {  
                return Int(Date().timeIntervalSinceDate(start))  
            } else {  
                return 0  
            }  
        }  
        /* Også mulig å sette, om ønskelig:  
        set(newValue) {...}  
        */  
    }  
}
```

```
let server = Server(ip: "192.168.0.1")  
server.boot()  
Thread.sleepForTimeInterval(5)  
print("Up for \(server.uptime) seconds")
```

# Property observers

```
class Server {  
    var ip: String {  
        willSet(newIp) {  
            print("Kommer til å sette ip til \$(newIp)")  
        }  
        didSet {  
            print("Satte ip til \$(ip)")  
        }  
    }  
}
```

# **Type properties/type methods - (aka static)**

Opererer på typenivå (klasse/struct), uten at man trenger en instans

# Eksempel for klasser

```
class KlasseUtils {  
    class var typeProperty: Int {  
        get {  
            return 1  
        }  
    }  
    class func typeMethod() {}  
}
```

KlasseUtils.typeProperty

KlasseUtils.typeMethod()

# Eksempel for structs

```
struct StructUtils {  
    static var typeProperty: Int = 0  
    static func typeMethod() {}  
}
```

StructUtils.typeProperty

StructUtils.typeMethod()

# Access control

- Swift defaulter til fornuftig access control, derfor ikke alltid nødvendig å tenke på dette
- Blir viktig når man lager frameworks
- Som standard internal
- Tips: sett metoder private som default

```
class SomeInternalClass {}           // implisitt internal
var someInternalConstant = 0         // implisitt internal
```



# Nivåer

## Kortversjonen

- **private** - skjules utad unntatt i deklarasjonen
- **fileprivate** - skjules utenfor filen
- **internal** - tilgjengelig i samme modul
- **public** - tilgjengelig også for andre moduler
- **open** - tilgjengelig også for andre moduler

# Videre lesning

- side 12-29 TSPL
- <http://goshdarnclosuresyntax.com/>

## Oppgaver

**Se Øvingsoppgaver på IT's learning /  
GitHub**