Closures

```
// Syntax
{ (parameters) -> return type in
    statements
}
```

1. Closures are self-contained blocks of functionality that can be passed around and used in your code.

Closures Syntax Optimizations

```
let names = ["Chris", "Alex", "Ewa", "Barry",
"Daniella"]
// 1
revers edNames = names.s or ted(by: { (s1: String,
s2: String) -> Bool in
        return s1 > s2
})
// 2. Inferring Type From Context
revers edNames = names.s or ted(by: { s1, s2 in
return s1 > s2 } )
// 3. Implicit Returns from Single -Ex pre ssion
Closures
revers edNames = names.s or ted(by: { s1, s2 in s1
> s2 } )
// 4. Shorthand Argument Names
revers edNames = names.s or ted(by: { $0 > $1 } )
// 5. Operator Methods, operator (>) is a method
revers edNames = names.s or ted(by: >)
// 6. Trailing Closures
revers edNames = names.s or ted() { $0 > $1 }
// function with only one arguement
revers edNames = names.s orted { $0 > $1 }
```

1. A trailing closure is written after the function call's parentheses, even though it is still an argument to the function.

Escaping Closures

```
var completionHandlers: [() -> Void] = []
func someFu nct ion Wit hEs cap ing Clo sur e(h -
andler: @escaping () -> Void) {
        com ple tio nHa ndl ers.ap pen d(h andler)
}
func someFu nct ion Wit hNo nes cap ing Clo sur -
e(c losure: () -> Void) {
```

Escaping Closures (cont)

```
>    closure()
}
class SomeClass {
    var x = 10
    func doSomething() {
        someFunctionWithEscapingClosure {
            self.x = 100
        }
        someFunctionWithNonescapingClosure {
            x = 200
        }
    }
}
```

1. A closure is said to escape a function when the closure is passed as an argument to the function, but is called after the function returns.
2. Marking a closure with *@escaping* means you have to refer to self explicitly within the closure.
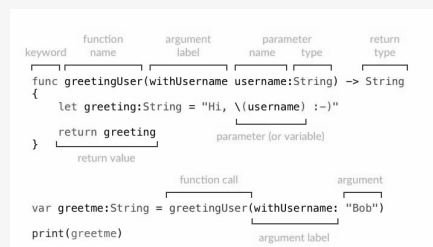
Autoclosures

```
var customersInLine = ["Chris", "Alex", "Ewa",
"Barry", "Daniella"]
func serve( cus tomer custom erP rov ider:
@autoc losure () -> String) {
        pri nt( "Now serving \(cust ome rPr ovi -
der ()) !")
}
serve( cus tomer: custom ers InL ine.re mov e(at:
0))
```

1. An *autoclosure* is a closure that is automatically created to wrap an expression that's being passed as an argument to a function.
2. An autoclosure lets you delay evaluation.

Functions


```
        function    argument    parameter    return
keyword   name      label        name  type   type
func greetingUser(withUsername username:String) -> String
{
    let greeting:String = "Hi, \(username) :-)"
}   return greeting              parameter (or variable)
        return value
                    function call           argument
var greetme:String = greetingUser(withUsername: "Bob")
print(greetme)
                                 argument label
```

1. Functions are self-contained chunks of code that perform a specific task.

By **zyqhi**
cheatography.com/zyqhi/

Not published yet.
Last updated 10th April, 2019.
Page 1 of 2.

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
http://crosswordcheats.com

Property

```
class SomeClass {
        // Type Property
        static var stored Typ ePr operty = "Some
value."
        static var comput edT ype Pro perty: Int {
                return 27
        }
        class var overri dea ble Com put edT ype -
Pro perty: Int {
                return 107
        }

        // Lazy Stored Property
        lazy var view = UIView()

        // Stored Property
        let data = [String]()

        // Computed Property
        var id: Int = 0 {
                willSet {
                        pri nt( " About to set \
(newV alu e)")
                }
                didSet {
                        pri nt( "Did set id to \
(id), oldValue is \(oldV alu e)")
                }
        }

        var idStr: String {
                set {
                        id = Int(idStr) ?? 0
                }
                get {
                        return String(id)
                }
        }

        var desc: String {
                return " Rea donly proper ty."
        }
```

Property (cont)

```
>}
```

1. Computed properties must be declared with the *var* keyword, because their value is not fixed.

2. A lazy stored property is a property whose initial value is not calculated until the first time it is used.

Enumeration

```
/// Use enum
enum Planet {
        case mercury, venus, earth, mars, jupiter,
saturn, uranus, neptune
}
var p = Planet.venus
p = .uranus
/// Iterating over Enumer ation Cases
enum Beverage: CaseIt erable {
        case coffee, tea, juice
}
let number OfC hoices = Bevera ge.a ll Cas es.c -
ount
print( " \(n umb erO fCh oices) beverages availa -
ble ")
for beverage in Bevera ge.a ll Cases {
        pri nt( " bev ergae is \(beve rag e)")
}
/// Associated Values
enum Barcode {
        case upc(Int, Int, Int, Int)
        case qrCode (St ring)
}
```

1. An enumeration defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code.

2. Conforming to the *CaseIterable* protocol makes a enum iterable.

By **zyqhi**
cheatography.com/zyqhi/

Not published yet.
Last updated 10th April, 2019.
Page 2 of 2.

Sponsored by CrosswordCheats.com
Learn to solve cryptic crosswords!
http://crosswordcheats.com