

# Objective-C Basis

23 april 2005, Eindhoven

© Patrick Machielse

[patrick@hieper.nl](mailto:patrick@hieper.nl)

# Algemeen

```
// extensies
```

Objective-C code in bestanden met .m extensie

```
// commentaar moet!
```

```
/* Alles hiertussen wordt genegeerd */
```

```
// Alles tot het einde van deze regel ook!
```

```
// Namen van variabelen beginnen met kleine
```

```
// letter. Woorden worden aangegeven door
```

```
// hoofdletters. Begin nooit met een _!
```

```
goedeVariableNaam
```

```
_slechtenaam
```

# Data typen

```
// Data typen uit C
```

```
char          'a'          unsigned  
int           12           short, long  
float         3.14  
double        2.71         long
```

```
// Objective-C toevoegingen
```

```
BOOL          YES of NO  
id            een willekeurig Cocoa 'object'
```

# Operators (1)

```
// voor de hand liggend
```

```
+, -, /, *, =
```

```
// pre- en postfix
```

```
int i = 0, rij[] = {1, 2, 3};
```

```
int j = rij[++i]; // 2!
```

```
int j = rij[i++]; // 1!
```

```
// korte schrijfwijze
```

```
j = j + 12;
```

```
j += 12;
```

## Operators (2)

```
// modulo
```

```
int m = 13 % 4; // 1
```

```
// boolean operators && (en) en || (of)
```

```
BOOL waar = YES && YES;
```

```
BOOL waar = YES || NO;
```

```
BOOL onwaar = YES && NO;
```

```
BOOL onwaar = NO || NO;
```

```
// vergelijken == (gelijk aan) != (ongelijk)
```

```
BOOL even = ((21 % 2) == 0)
```

# Arrays

```
// Een array is een rij datatypen
```

```
int intArray[3];
```

```
// Eerste element heeft index 0
```

```
intArray[0] = 1;
```

```
// Automatische initialisatie
```

```
int intArray[] = {1, 2, 3};
```

```
// String is char array met '\0' terminatie
```

```
char woord[] = "VAMP";
```

```
char woord[] = {'V', 'A', 'M', 'P', '\0'};
```

## Pointers (\* VERWARREND! \*)

```
// Een pointer is een 'geheugenadres'  
// '*' in declaratie: 'is een pointer'  
int *p;  
  
// '&' neem adres van  
int twaalf = 12;  
p = &twaalf;  
int *dozijn = &twaalf;  
  
// '*' in expressie: 'de waarde van'  
// in C spraak 'dereference the pointer'  
*p = 13; (of: *dozijn = 13;)
```

# Pointers en Arrays

```
// Array naam == pointer naar eerste element  
int rij[] = {1, 2, 3};  
int *een = rij;  
*een = 4; // rij[0] heeft nu waarde 4...
```

```
// Strings zijn ook pointers  
char vamp[] = "VAMP";  
char *vamp = "VAMP";  
char vee = *vamp; // vee heeft waarde 'V'
```

# Loops

```
// 'for' loop  
for ( a = 0; a < 10; a++ ) { doe iets; }
```

```
// 'while' loop  
// functioneel gelijk aan 'for' loop  
while ( a < 10 ) { doe iets; }
```

```
// 'do while' loop  
// minstens 1 maal uitgevoerd  
do { doe iets; }  
while ( a < 10 )
```

# If Else

```
// kiest 1 van de volgende opties
// else en else if optioneel
if ( a == 0 ) {
    // doe iets als a == 0
}
else if ( a == 1 ) {
    // doe iets anders als a == 1
}
else {
    // nog iets anders
    // als a != 0 en a != 1
}
```

# Switch

```
// spring direct naar een optie
// a moet een integer zijn!
switch ( a ) {
    case 0:
        // doe iets als a == 0
        break;
    case 1:
        // doe iets als a == 1
        break;
    default:
        // nog iets anders
        // als a != 0 en a != 1
}
```

# Funcities

```
// C functies hebben de volgende vorm
returType fuctieNaam(argType argNaam, ...)

// bijvoorbeeld:
float oppervlak(float lengte, float breedte) {
    float opp = lengte * breedte;
    return opp;
}
float o = oppervlak(12.0, 23.4);

// de code tussen {} is de 'body'.
// 'opp' is niet geldig buiten de functie
// return beëindigt de functie
```

# Hallo VAMP

```
#import <Foundation/Foundation.h>

// startpunt voor *alle* programma's
int main(int argc, char *argv[])
{
    // print naar de console
    NSLog(@"Hallo VAMP");

    // 'exit value' 0: geen fouten
    return 0;
}
```

**Objecten, levenscyclus, geheugenbeheer**

## Message passing

```
// centrale paradigma van Objective-C
// roep een methode zo aan:
[receiver message];

// receiver is een Objective-C object
[NSArray array]; // class object
[@"VAMP" length]; // instance object

// nil is een speciale constante, 'null object'
// je mag methoden aanroepen van een nil object
// met als resultaat nil, _geen_ foutmelding!
[nil eenOfAnderereMethode];
```

## Methoden (1)

```
// Een methode is een 'functie' die bij een  
// object hoort. Instance methoden hebben  
// deze vorm:
```

```
-(returnType)methodeNaam {}
```

```
// Class methoden beginnen met een '+' teken
```

```
+(returnType)methodeNaam {}
```

```
// Argumenten worden gescheiden door ':' en  
// hebben een label/naam
```

```
-(returnType)doeIetsMet:(argType argNaam) {}
```

## Methoden (2)

```
//  bijvoorbeeld:
+(float)oppervlakMetLengte:(float)lengte
                          enBreedte:(float)breedte
{
    return lengte * breedte;
}
float o = [MijnCirkel oppervlakMetLengte:12.0
          enBreedte:3.0];

//  de naam van een methode (inclusief :)
//  'heet' een _selector_
```

# Object Levenscyclus

- Creëer een object
  - maak zelf een nieuw object
- Gebruik een object
  - stuur er messages naartoe
  - [mijnObject doeIets];
- Een object wordt afgedankt als het niet meer wordt gebruikt.

# Geheugenbeheer in een notedop

- Geen automatisch geheugenbeheer
- Ieder object heeft een 'retain count'
- Verhoog de retain count met:
  - alloc
  - new
  - retain
- Verlaag de retain count met:
  - release
  - autorelease
- Als de retain count de waarde 0 bereikt wordt de 'dealloc' methode aangeroepen, en wordt het object vernietigd

## Objecten (instances) maken

```
// -alloc reserveert geheugen,  
// -init initialiseert nieuwe object  
MijnClass *mijnObject;  
mijnObject = [[MijnClass alloc] init];  
  
// ook wel: -new, maar bijna nooit gebruikt  
mijnObject = [MijnClass new];  
  
// niet goed is:  
mijnObject = [MijnClass alloc];  
[mijnObject init];  
// -init retourneert mogelijk een ander object  
// dan alloc!
```

## Instances afdanken

```
// stuur -release of -autorelease
// als je geen interesse meer hebt in
[mijnObject release];

// dit verlaagt de retain count met 1

// als de retain count gelijk is aan 0
// wordt automatisch de -dealloc methode van
// het object aangeroepen
```

## -release en -autorelease

```
// -release verlaagt de retain count van een  
// object _nu_. Na -release kan het object  
// vernietigd zijn.
```

```
// -autorelease verlaagt de retain count  
// _later_. Binnen de huidige functie of  
// methode blijft het object intact.
```

```
// -autorelease vereist het bestaan van een  
// NSAutoreleasePool object:  
[[NSAutoreleasePool alloc] init];
```