Telerik Academy

Objective-C Fundamentals

The Basic Concepts in Obj-C

Mobile apps for iPhone & iPad

Telerik Software Academy
http://academy.telerik.com

Telerik Academy

Table of Contents

Telerik Academy

Data Types and Variables

Telerik Academy

Data Types and Variables

- Objective-C supports all of the standard primitive types
  - Integer and floating-point numbers
  - Characters and Strings
  - Something like a Boolean type

Telerik Academy

Number Types

Integer and Floating-point

Telerik Academy

Number Types in Objective-C

- Objective-C supports different types of numbers
  - Using modifiers to change the range of values
- Types:
  - `char`, `int`, `double` and `float`
- Modifiers:
  - unsigned, long, short and long long
- Apple suggests using their typedefs
  - NSInteger, NSUInteger, CGFloat, etc…

Telerik Academy

Number Types

Live Demo

Telerik Academy

Boolean Type

Boolean Type

- Objective-C has no true Boolean (Yes/No)
  - It has Yes-like and No-like values
  - The values `nil`, `Null`, `0` and `@""` are No-like
  - Everything else is Yes-like
    - `1`, `2`, `1.2`, `@"any string"`, etc…
- There is a BOOL type that is just a `typedef`:

```
typedef signed char BOOL;
#define OBJC_BOOL_DEFINED
#define YES (BOOL)1
#define NO (BOOL)0
```

  - i.e. BOOL is just a signed char

Telerik Academy

Boolean Type

Live Demo

NSNumber

- NSNumber is a type that holds numbers
  - Any number from short to int to unsigned long
  - It is a object type
  - The only way to put a number into a collection
    - NSArray, NSDictionary, etc…

```
NSNumber *intNumber = @123; //NSNumber with int
NSNumber *longNumber = @123l; //NSNumber with long
NSNumber *doubleNumber = @123.3;
NSNumber *boolNumber = @YES;
```

Telerik Academy

NSNumber

Live Demo

Telerik Academy

The String Type

Telerik Academy

The String Type

- To define a string use the `NSString` type
  - Has a literal to set its value directly
    - Using the `"string value"` and a `prefix @`

```
NSString *str = @"Literal";
```

- The string is immutable type
  - i.e. once set, it cannot be changed

Telerik Academy

The Mutable String

- To create a mutable string, use the `NSMutableString` type
  - Creates a string that can be changed
  - Created as a regular string – using the literal
  - Has methods for appending string at the end

```
NSMutableString *str = @"";
[str appendString:"new str"];
```

Telerik Academy

Strings

Live Demo

Telerik Academy

Strings Concatenation

- Strings can be concatenated
  - The result is the second string appended at the end of the first string
- Concatenation is done using the methods:
  - `stringByAppendingString: (NSString*)]`
  - `stringByAppendingFormat: (NSString*), …]`
- Since `NSString` is immutable, both return a new string object

Telerik Academy

Concatenating Strings

Live Demo

Telerik Academy

Slow String Concatenation

- Concatenating strings using the `stringByAppending*` is very slow
  - For every concatenation, a new string object is allocated and initialized in memory
  - Since `NSString` is an immutable, n objects are allocated, then deallocated, for concatenating n strings
- When in need of multiple string concatenations use `NSMutableString`
  - Has methods `appendString*: (NSString*)`
  - Allocates only `O(log2(n))` objects

Telerik Academy

Faster String Concatenation

Live Demo

Telerik Academy

Conditional Statements

Make the Code Run Conditionally

Telerik Academy

Conditional Statements

- Objective-C supports the `if-else` construction

```
if(condition)
{
  //run code
}
else if(second-condition)
{
  //run other code
}
else
{
  //run third code
}
```

- The condition can be of any type
  - `nil`, `NO` and `0` are evaluated as false conditions

Telerik Academy

Conditional Statements

- Objective-C supports the `if-else` construction

```
if(condition)
{
  //run code
}
else if(second-condition)
{
  //run other code
}
else
{
  //run third code
}
```

Evaluated if the first condition is false

- The condition can be of any type
  - `nil`, `NO` and `0` are evaluated as false conditions

Telerik Academy

Conditional Statements

- Objective-C supports the `if-else` construction

```
if(condition)
{
  //run code
}
else if(second-condition)
{
  //run other code
}
else
{
  //run third code
}
```

Evaluated if the first condition is false

Executed if all the conditions are false

- The condition can be of any type
  - `nil`, `NO` and `0` are evaluated as false conditions

Telerik Academy

Conditionals: if-else

Live Demo

Telerik Academy

Switch-Case Construction

- Switch-case is a construction similar if-else
  - Checks the value of a given object
    - The object must be either number, BOOL or enum
  - If the object has a value equal to any of the cases the code in this case is executed

```
switch(choice)
{
  case 1: // the code for this case
  break;
  case 2: // the code for this case
  break;
  default: //default behavior
  break;
}
```

Telerik Academy

Switch-Case Construction

- Switch-case is a construction similar if-else
  - Checks the value of a given object
    - The object must be either number, BOOL or enum
  - If the object has a value equal to any of the cases the code in this case is executed

```
switch(choice)
{
  case 1: // the code for this case
  break;
  case 2: // the code for this case
  break;
  default: //default behavior
  break;
}
```

Missing break applies the fall-thought rule

Telerik Academy

Switch-Case Construction

- Switch-case is a construction similar if-else
  - Checks the value of a given object
    - The object must be either number, BOOL or enum
  - If the object has a value equal to any of the cases the code in this case is executed

```
switch(choice)
{
  case 1: // the code for this case
  break;
  case 2: // the code for this case
  break;
  default: //default behavior
  break;
}
```

Missing break applies the fall-thought rule

Acts like the final else in if-else construction

Telerik Academy

Conditionals: Switch-case

Live Demo

Telerik Academy

Loops

Create repeating parts of code

Telerik Academy

Loops in Objective-C

- Loops enable the execution of code multiple times
  - The number of loop iterations can be exact or while a condition is fulfilled
- Objective-C supports four kinds of loops:
  - for – makes an exact number of iterations
  - for-in – iterates through a collection of objects
  - while – runs the code until a condition is fulfilled
  - do-while – the same as while

For Loop

- for runs a block of code a number of times

```
for(initialization; condition; increment)
{
  //execute code;
}
```

For Loop

- for runs a block of code a number of times

```
for(initialization; condition; increment)
{
  //execute code;
}
```

  - Initialization is used to set the initial state
    - i.e. set the start of a counter

For Loop

- for runs a block of code a number of times

```
for(initialization; condition; increment)
{
  //execute code;
}
```

  - Initialization is used to set the initial state
    - i.e. set the start of a counter
  - Condition checks if the state is still valid and whether the loop should continue or stop

For Loop

- for runs a block of code a number of times

```
for(initialization; condition; increment)
{
  //execute code;
}
```

  - Initialization is used to set the initial state
    - i.e. set the start of a counter
  - Condition checks if the state is still valid and whether the loop should continue or stop
  - Increment means update
    - Change the state of the loop

Telerik Academy

For Loop

Live Demo

Telerik Academy

For-in Loop

- for-in is used to iterate over collection of objects

```
for(type *obj in collection)
{
  //code
}
```

  - Iterates through the objects stored in a collection
  - Can be used only with collections
    - NSArray, NSDictionary, NSSet, etc..
  - obj gets all the values from the collection

Telerik Academy

For-in Loop

Live Demo

While Loop

- The while loop executes a piece of code, while a given condition is fulfilled

```
while(condition)
{
  //code
}
```

  - Condition is evaluated at every iteration of the loop
    - If it is YES, the loop continues execution
    - If it is NO, the loop stops its execution
  - If the condition has an initial value NO the code in the loop is not executed

Telerik Academy

While Loop

Live Demo

Do-while Loop

- The do-while loop is pretty much like while
  - Has a slightly different construction
  - If the initial value of the condition is NO, the code is executed exactly once

```
do {
  //code
} while(condition)
```

Telerik Academy

Do-while Loop

Live Demo

Nesting Loops

- Loops can be nested inside one another
  - The code inside a loop block can contain indefinite number of other loops

```
for(…)
{
  while(…)
  {
    //code
  }
  for(id value in array)
  {
    //code
  }
}
```

Telerik Academy

Nested Loops

Live Demo

Telerik Academy

Arrays

Arrays

- Arrays are objects that hold other objects
  - Also known as collections
- Each object can be accessed via an index
  - Indexes range from `0` to the number of objects in the array minus `1`
- Arrays can hold objects of any type
  - `Numbers` must be wrapped into `NSNumber`

Arrays (2)

- Many ways to create arrays
  - Most common is:

```
NSArray *arr = @["one", @"two", @"three", nil];
```

  - NSArray is immutable
    - i.e. it cannot be changed
- To create a changeable array use NSMutableArray

Telerik Academy

Creating Arrays

Live Demo

Telerik Academy

Functions

Telerik Academy

Functions in Objective-C

- Functions are named pieces of code
  - The code inside a function can be executed using the function identifier

```
int max(int n1, int n2)
{
  if(n1 > n2) return n1;
  else return n2;
}
int main()
{
  NSLog(@"%d", max(1,5))
}
```

Telerik Academy

Functions in Objective-C

- Functions are named pieces of code
  - The code inside a function can be executed using the function identifier

```
int max(int n1, int n2)
{
  if(n1 > n2) return n1;
  else return n2;
}
int main()
{
  NSLog(@"%d", max(1,5))
}
```

Identifier: max

Telerik Academy

Functions in Objective-C

- Functions are named pieces of code
  - The code inside a function can be executed using the function identifier

```
int max(int n1, int n2)
{
  if(n1 > n2) return n1;
  else return n2;
}
int main()
{
  NSLog(@"%d", max(1,5))
}
```

Identifier: max

Return type: int

Telerik Academy

Functions in Objective-C

- Functions are named pieces of code
  - The code inside a function can be executed using the function identifier

```
int max(int n1, int n2)
{
  if(n1 > n2) return n1;
  else return n2;
}
int main()
{
  NSLog(@"%d", max(1,5))
}
```

Identifier: max

Return type: int

Parameters: n1 and n2

Telerik Academy

Functions in Objective-C

- Functions are named pieces of code
  - The code inside a function can be executed using the function identifier

```
int max(int n1, int n2)
{
  if(n1 > n2) return n1;
  else return n2;
}
int main()
{
  NSLog(@"%d", max(1,5))
}
```

Identifier: max

Return type: int

Parameters: n1 and n2

Signature: int(int, int)

Telerik Academy

Functions in Objective-C (2)

- Every function has the following:
  - Identifier: the name of the function
  - Return type: int, NSArray, id or void
  - Parameters: 0 or more parameters passed to the function
- Returning void means that the function returns no result
  - And the caller of the function shouldn't expect any

Telerik Academy

Functions

Live Demo

Telerik Academy

Objective-C Fundamentals

Questions?

http://academy.telerik.com

Homework

1. Create a matrix (array of arrays) that contains integer values using the schema and print them to the console

1	2	3
8	9	4
7	6	5

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

2. Create a function that generates the above matrix by a given N