# Blocks

* It helps keep the method call and callback code together.
* These are anonymous functions.
* These are inline functions.
* Blocks are by default saved on the heap.
* You can access both stack and heap variables.
* An Objective-C class defines an object that combines data with related behavior. Sometimes, it makes sense just to represent a single task or unit of behavior, rather than a collection of methods.
* Blocks are a language-level feature added to C, Objective-C and C++, which allow you to create distinct segments of code that can be passed around to methods or functions as if they were values. Blocks are Objective-C objects, which means they can be added to collections like NSArray or NSDictionary. They also have the ability to capture values from the enclosing scope, making them similar to *closures* or *lambdas* in other programming languages.
* Syntax:
  + [myObject doSomething:^(MyObject \*object){
  + // callback code...
  + }];
* Example:
  + [videoGames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL \*stop) {
  + NSLog(@"Video game: %@", (NSString \*)obj);
  + }];
  + // Here lets say videoGames is an NSArray and we use the block to print it one by one.
* Saving a block in a property than can be used later:
  + @property (nonatomic, copy) void (^block)(id, NSUInteger, Bool\*);
  + //Blocks by default are saved on the heap. If you want to use it later then you have to save it one the heap and hence copy is used.
  + // Here a block of code was created
  + [self doSomethingWithBlock:^(id, NSUInteger, BOOL \*) {
  + NSLog(@"Done!");
  + }];
  + // Here the block was saved in the property
  + -(void)doSomethingWithBlock:(void (^)(id, NSUInteger, BOOL\*))block {
  + self.block = block;
  + }
  + // The block was then called.
  + -(void)afterOneSecond {
  + BOOL stop;
  + self.block(@"The Legend of Zelda", 0, &stop);
  + }
  + // so in this example we saved a block for future use.
* The best that about blocks is that it can access variables that are outside the scope of current block. You can access the following things:
  + global variables obviously
  + parameters passed to the block obviously
  + stack variables(takes a copy of the value)
  + \_block variables(provided by reference)
  + any local variables in the block obviously
* Example of accessing stack variables
  + NSSTring \*favouriteGame = @"Fallout 2";
  + [videoGames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL \*stop) {
  + NSString \*game = (NSString \*)obj;
  + if ([game isEqualToString:favouriteGame]) {
  + NSLog(@"Woot, %@ is my favourite game!", game);
  + }
  + else {
  + NSLog(@"Video game: %@", (NSString \*)obj);
  + }
  + }];
* Example of variables passed using reference
  + int countOfGames = 0;
  + NSSTring \*favouriteGame = @"Fallout 2";
  + [videoGames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL \*stop) {
  + NSString \*game = (NSString \*)obj;
  + if ([game isEqualToString:favouriteGame]) {
  + NSLog(@"Woot, %@ is my favourite game!", game);
  + }
  + else {
  + NSLog(@"Video game: %@", (NSString \*)obj);
  + }
  + countOfGames++;
  + }];
  + // This won’t work because it’s only a copy. You can only use this for reading purposes
  + // If you want something for writing purposes then you have to move the variable to heap using the \_block keyword.
  + \_\_block int countOfGames = 0;
  + NSSTring \*favouriteGame = @"Fallout 2";
  + [videoGames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL \*stop) {
  + NSString \*game = (NSString \*)obj;
  + if ([game isEqualToString:favouriteGame]) {
  + NSLog(@"Woot, %@ is my favourite game!", game);
  + }
  + else {
  + NSLog(@"Video game: %@", (NSString \*)obj);
  + }
  + countOfGames++;
  + }];
* You have to be careful that this can also cause retain cycles.