

Making Things Modular

INTRODUCTION



Simon Allardice

STAFF AUTHOR, PLURALSIGHT

@allardice www.pluralsight.com

All Grouping is Separation

All Grouping is Separation



All Grouping is Separation



ch. 1



ch. 2



ch. 3

All Grouping is Separation



ch. 1



ch. 2



ch. 3



All Grouping is Separation



ch. 1

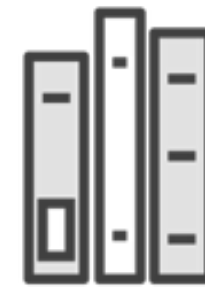


ch. 2



ch. 3

All Grouping is Separation



ch. 1



ch. 2



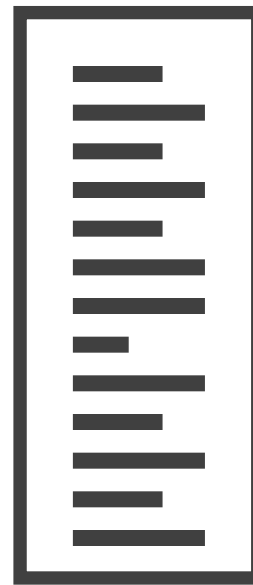
ch. 3

Breaking Code Apart

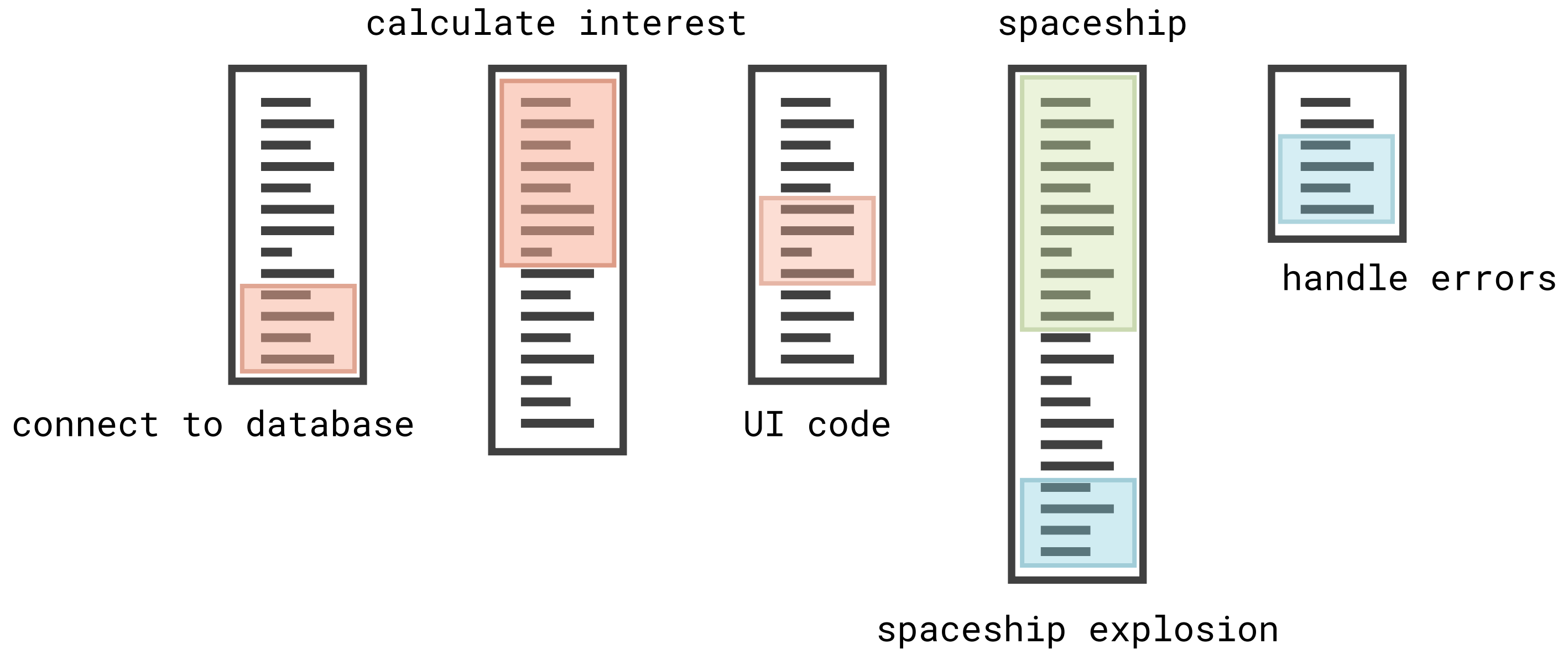
Breaking Code Apart



Breaking Code Apart



Breaking Code Apart



Summary

Summary

Summary

Functions

Summary

Functions

Return Types / Parameters

Summary

Functions

Return Types / Parameters

Recursion

Summary

Functions

Return Types / Parameters

Recursion

Composite Data Types

Summary

Functions

Return Types / Parameters

Recursion

Composite Data Types

Arrays and Collections

Modular Code: Creating Functions

Making Functions

// more above

// more below

Making Functions

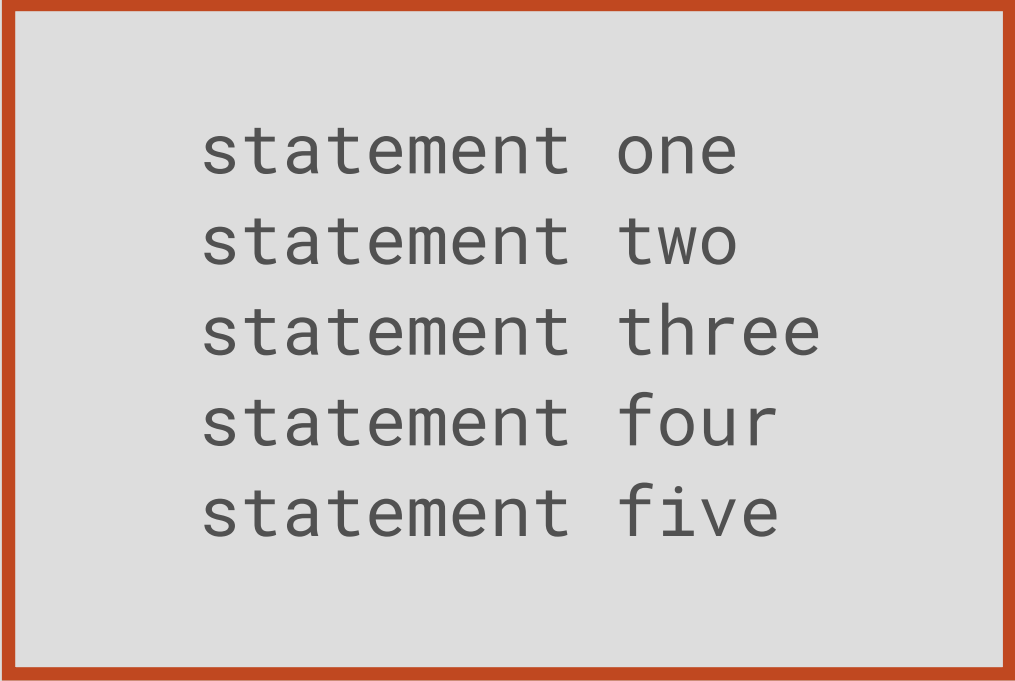
```
// more above
```

```
statement one  
statement two  
statement three  
statement four  
statement five
```

```
// more below
```

Making Functions

// more above



```
statement one  
statement two  
statement three  
statement four  
statement five
```

// more below

Making Functions

```
// more above
```

```
statement one  
statement two  
statement three  
statement four  
statement five
```

```
// more below
```

Making Functions

// more above

```
Function myNewFunction  
statement one  
statement two  
statement three  
statement four  
statement five  
End Function
```

// more below

Making Functions

// more above

```
Function validatePassword  
statement one  
statement two  
statement three  
statement four  
statement five  
End Function
```

// more below

Making Functions

// more above

```
Function playSoundEffect  
statement one  
statement two  
statement three  
statement four  
statement five  
End Function
```

// more below

Making Functions

// more above

```
Function showScoreboard  
statement one  
statement two  
statement three  
statement four  
statement five  
End Function
```

// more below

Making Functions

// more above

```
Function explodeSpaceship  
statement one  
statement two  
statement three  
statement four  
statement five  
End Function
```

// more below

Making Functions

// more above

Function createEmail

statement one

statement two

statement three

statement four

statement five

End Function

// more below

Making Functions

// more above

Function encryptFile

statement one

statement two

statement three

statement four

statement five

End Function

// more below

Making Functions

// more above

Function connectToDatabase

statement one

statement two

statement three

statement four

statement five

End Function

// more below

Making Functions

// more above

Function printMessage

statement one

statement two

statement three

statement four

statement five

End Function

// more below

Making Functions

```
// more above
```

```
function printMessage {  
statement one  
statement two  
statement three  
statement four  
statement five  
}
```

```
// more below
```

Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

```
// more below
```

Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

```
// more below
```

```
// call the function
```

```
printMessage()
```

Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

```
// more below
```

```
// call the function
```

```
printMessage()
```

Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

```
// more below
```

```
// call the function
```

```
printMessage()
```

Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

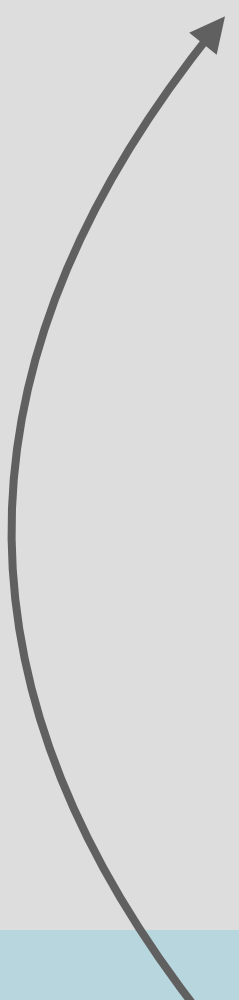
```
// more below
```

```
// call the function
```

```
printMessage()
```

Making Functions

// more above



```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

// more below

// call the function

```
printMessage()
```

Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

```
// more below
```

```
// call the function
```

```
printMessage()
```


Making Functions

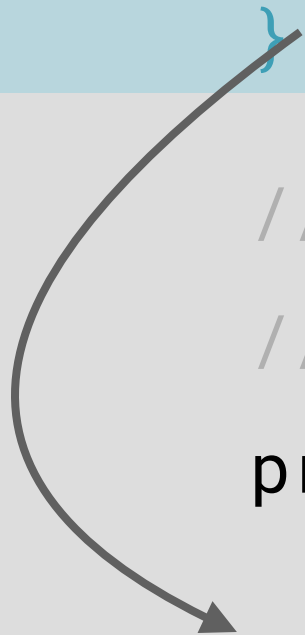
```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five
```

```
// more below
```

```
// call the function
```

```
printMessage()
```



Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

```
// more below
```

```
// call the function
```

```
printMessage()
```

Making Functions

```
// more above
```

```
function printMessage {  
    statement one  
    statement two  
    statement three  
    statement four  
    statement five  
}
```

```
// more below
```

```
// call the function
```

```
printMessage()
```

Modular Code: Returning Values

Functions Which Return Values

Functions Which Return Values

get user's first name

get today's date

figure out the day of week

```
string message = "Hi, \ (userName). Have a great \ (dayOfWeek) !"
```

Functions Which Return Values

```
function createMessage
{
    get user's first name
    get today's date
    figure out the day of week
    string message = "Hi, \ (userName). Have a great \ (dayOfWeek) !"
}
```

Functions Which Return Values

```
function createMessage
{
    get user's first name
    get today's date
    figure out the day of week
    string message = "Hi, \ (userName). Have a great \ (dayOfWeek) !"
    return message
}
```


Functions Which Return Values

```
function createMessage
{
    get user's first name
    get today's date
    figure out the day of week
    string message = "Hi, \ (userName). Have a great \ (dayOfWeek) !"
    return message
}

// call the function - and expect a result
createMessage()
```

Functions Which Return Values

```
function createMessage
{
    get user's first name
    get today's date
    figure out the day of week
    string message = "Hi, \ (userName). Have a great \ (dayOfWeek) !"
    return message
}

// call the function - and expect a result
string myResult = createMessage()
```

Functions Which Return Values

```
function createMessage
{
    get user's first name
    get today's date
    figure out the day of week
    string message = "Hi, \(userName). Have a great \(dayOfWeek)!"
    return message
}

// call the function - and expect a result
string myResult = createMessage()
```



Functions Which Return Values

```
function createMessage
{
    get user's first name
    get today's date
    figure out the day of week
    string message = "Hi, \ (userName). Have a great \ (dayOfWeek) !"
    return message
}

// call the function - and expect a result
string myResult = createMessage()
```

Functions With Parameters

Functions With Parameters

```
function displayAreaOfRectangle (int width, int height)
{
    int area = width * height
    string message = "The area is:" + area
    print(message)
}
```

Functions With Parameters

parameters

```
function displayAreaOfRectangle (int width, int height)
{
    int area = width * height
    string message = "The area is:" + area
    print(message)
}
```

Functions With Parameters

parameters

```
function displayAreaOfRectangle (int width, int height)
{
    int area = width * height
    string message = "The area is:" + area
    print(message)
}

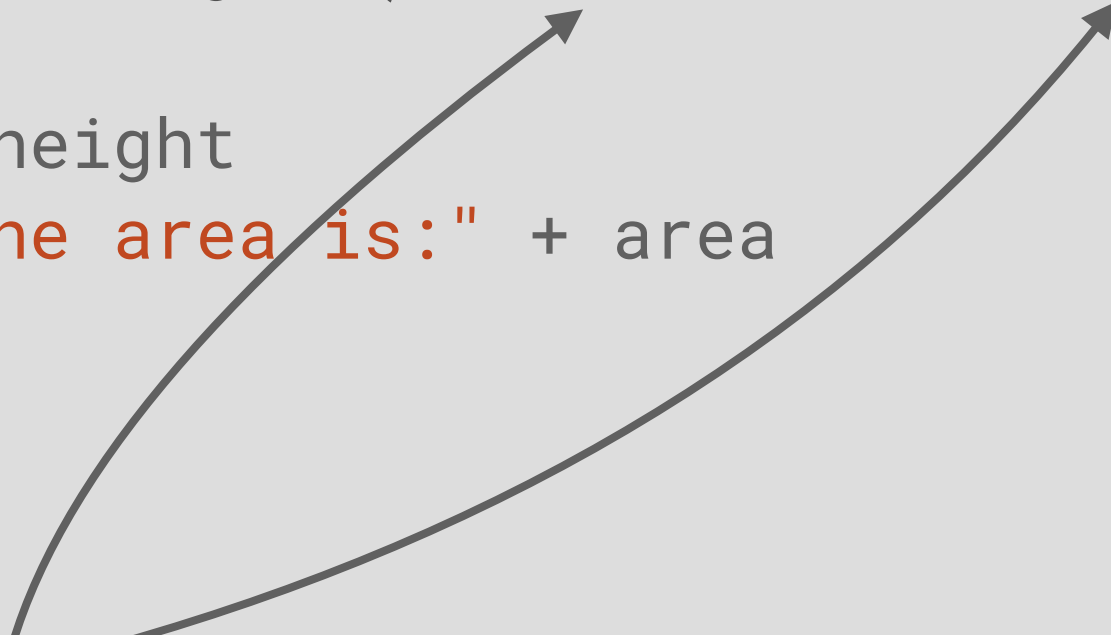
// call the function
displayAreaOfRectangle(6,10)
```


Functions With Parameters

parameters

```
function displayAreaOfRectangle (int width, int height)
{
    int area = width * height
    string message = "The area is:" + area
    print(message)
}

// call the function
displayAreaOfRectangle(6, 10)
```



The diagram consists of two curved arrows. The first arrow starts at the number '6' in the function call `displayAreaOfRectangle(6, 10)` and points to the `int width` parameter in the function definition `function displayAreaOfRectangle (int width, int height)`. The second arrow starts at the number '10' in the function call and points to the `int height` parameter in the function definition. This illustrates how arguments are passed to function parameters.

Functions With Parameters

parameters

```
function displayAreaOfRectangle (int width, int height)
{
    int area = width * height
    string message = "The area is:" + area
    print(message)
}
```

```
// call the function
```

```
displayAreaOfRectangle(6,10)    "The area is: 60"
```

Functions With Parameters

parameters

```
function displayAreaOfRectangle (int width, int height)
{
    int area = width * height
    string message = "The area is:" + area
    print(message)
}
```

// call the function

```
displayAreaOfRectangle(6, 10)    "The area is: 60"
displayAreaOfRectangle(5, 50)    "The area is: 250"
```

Functions With Parameters

parameters

```
function displayAreaOfRectangle (int width, int height)
{
    int area = width * height
    string message = "The area is:" + area
    print(message)
}
```

```
// call the function
```

```
displayAreaOfRectangle(6,10)    "The area is: 60"
```

```
displayAreaOfRectangle(5,50)    "The area is: 250"
```

```
displayAreaOfRectangle() // error - missing data!
```

Using Recursion

Introduction

```
function functionA {  
    // do stuff  
    // ...  
    functionB()  
    // ...  
}
```

Using Recursion

Introduction

```
function functionA {  
    // do stuff  
    // ...  
    functionB()  
    // ...  
}
```

```
// call function  
functionA()
```

Using Recursion

Introduction

```
function functionA {  
    // do stuff  
    // ...  
    functionB()  
    // ...  
}
```

```
// call function  
functionA()
```

Using Recursion

Introduction


```
function functionA {  
    // do stuff  
    // ...  
    functionB()  
    // ...  
}
```

```
// call function  
functionA()
```



Using Recursion

Introduction

```
function functionA {  
    // do stuff  
    // ...  
    functionB()  
    // ...  
}
```

```
// call function  
functionA()
```

```
function functionB {  
    // do other stuff  
    functionC()  
}
```

Using Recursion

Introduction

```
function functionA {  
  // do stuff  
  // ...  
  functionB()  
  // ...  
}  
  
// call functionA()  
functionA()
```

functionA calls functionB.

```
function functionB {  
  // do other stuff  
  functionC()  
}
```

functionB calls functionC.

```
function functionC {  
  // ...  
  functionD()  
  functionE()  
  functionF()  
  functionG()  
  // etc.  
}
```

The diagram illustrates a sequence of recursive calls: functionA calls functionB, and functionB calls functionC. The functionC definition is shown without further calls, indicating it is the base case or end of the sequence shown.

Using Recursion

Introduction

```
function functionA {  
    // do stuff  
    // ...  
    functionB()  
    // ...  
}  
  
// call functionA()  
functionA()
```

```
function functionB {  
    // do other stuff  
    functionC()  
}
```

```
function functionC {  
    // ...  
    functionD()  
    functionE()  
    functionF()  
    functionG()  
    // etc.  
}
```

The diagram illustrates recursive function calls. A call to `functionA()` leads to the execution of `functionA`, which calls `functionB()`. `functionB` then calls `functionC()`. `functionC` contains several other function definitions but no further calls are shown in this snippet. Arrows indicate the flow of control from the initial call to `functionA` through the sequence of recursive calls.

Using Recursion

Introduction

Recursive Function Calls

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

Recursive Function Calls

```
// pseudocode      one parameter - the name of a folder  
function listMP3Files (string currentFolder)  
    for each item in folder:currentFolder  
        if (item is mp3)  
            print(name of item)  
        end if  
    end for  
end function
```

Recursive Function Calls

```
// pseudocode      one parameter - the name of a folder  
function listMP3Files (string currentFolder)  
    for each item in folder:currentFolder  
        if (item is mp3)  
            print(name of item)  
        end if  
    end for  
end function
```


Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

Recursive Function Calls

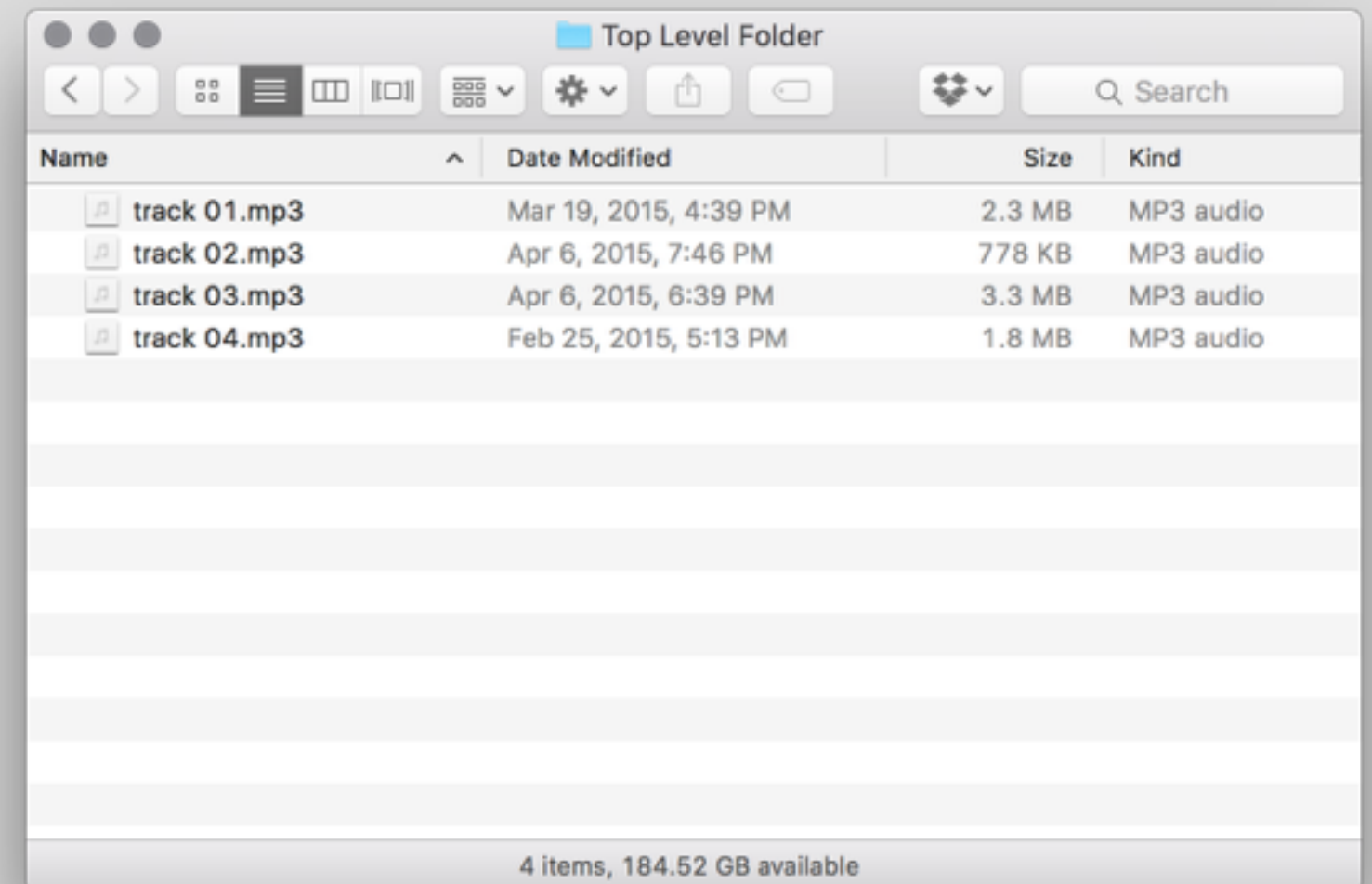
```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

```
// call the function
listMP3Files("/Top Level Folder")
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

```
// call the function
listMP3Files("/Top Level Folder")
```



Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

```
// call the function
listMP3Files("/Top Level Folder")
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

```
// call the function
listMP3Files("/Top Level Folder")
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
        if (item is a folder)
            for each sub-item in folder:item
                if (sub-item is mp3)
                    print(name of sub-item)
                end if
            end for
        end if
    end for
end function
```


Recursive Function Calls

// pseudocode

```
function listMP3Files (string currentFolder)
  for each item in folder:currentFolder
    if (item is mp3)
      print(name of item)
    end if
    if (item is a folder)
      for each sub-item in folder:item
        if (sub-item is mp3)
          print(name of sub-item)
        end if
      end for
    end if
  end for
end function
```

Recursive Function Calls

// pseudocode

```
function listMP3Files (string currentFolder)
  for each item in folder:currentFolder
    if (item is mp3)
      print(name of item)
    end if
    if (item is a folder)
      for each sub-item in folder:item
        if (sub-item is mp3)
          print(name of sub-item)
        end if
      end for
    end if
  end for
end function
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
        if (item is a folder)
            for each sub-item in folder:item
                if (sub-item is mp3)
                    print(name of sub-item)
                end if
            end for
        end if
    end for
end function
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
    end for
end function
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
        if (item is a folder)
            listMP3Files(name of item)
        end if
    end for
end function
```

Recursive Function Calls

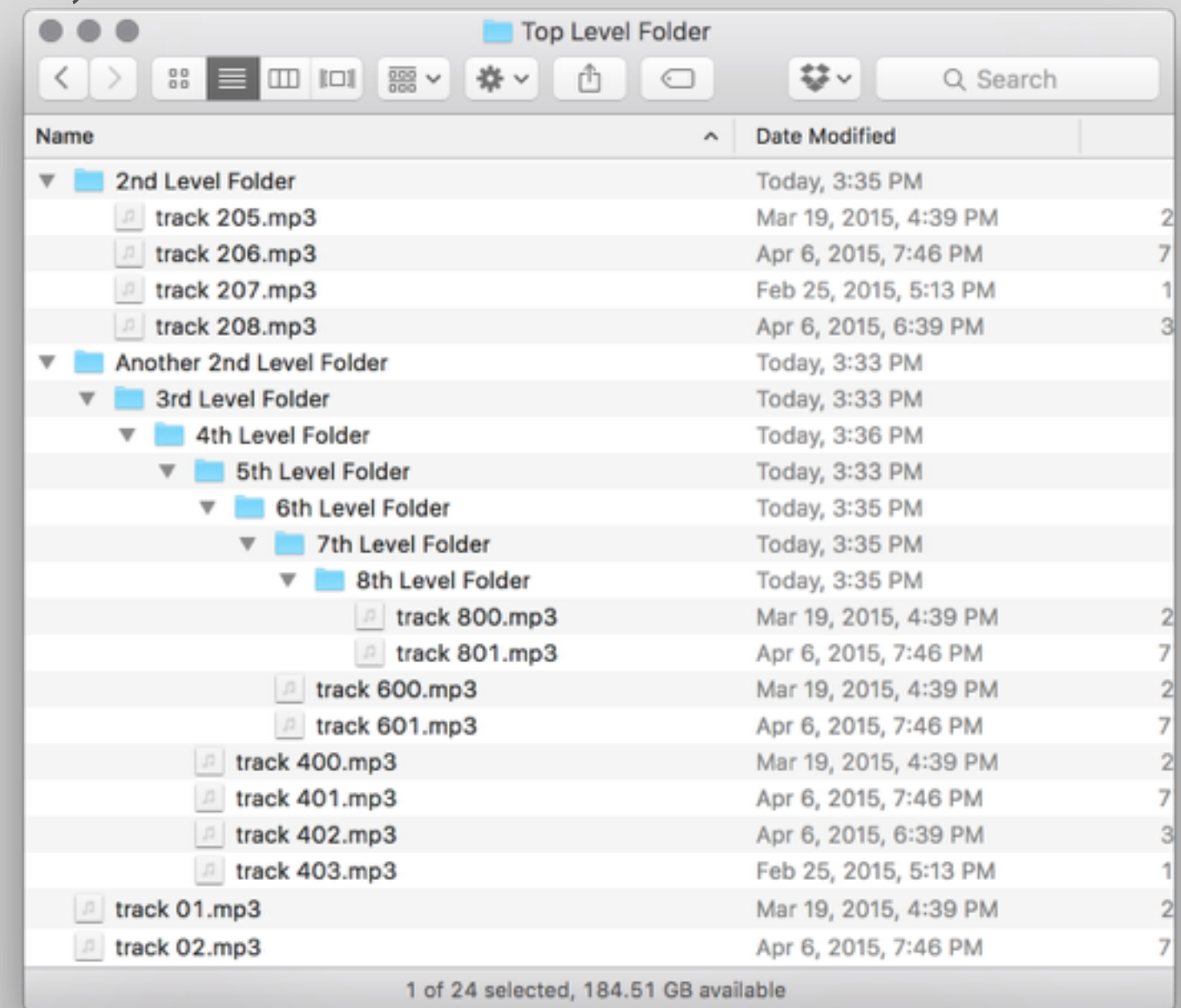
```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
        if (item is a folder)
            listMP3Files(name of item)
        end if
    end for
end function
```

```
// call the function
listMP3Files("/Top Level Folder")
```

Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
  for each item in folder:currentFolder
    if (item is mp3)
      print(name of item)
    end if
    if (item is a folder)
      listMP3Files(name of item)
    end if
  end for
end function

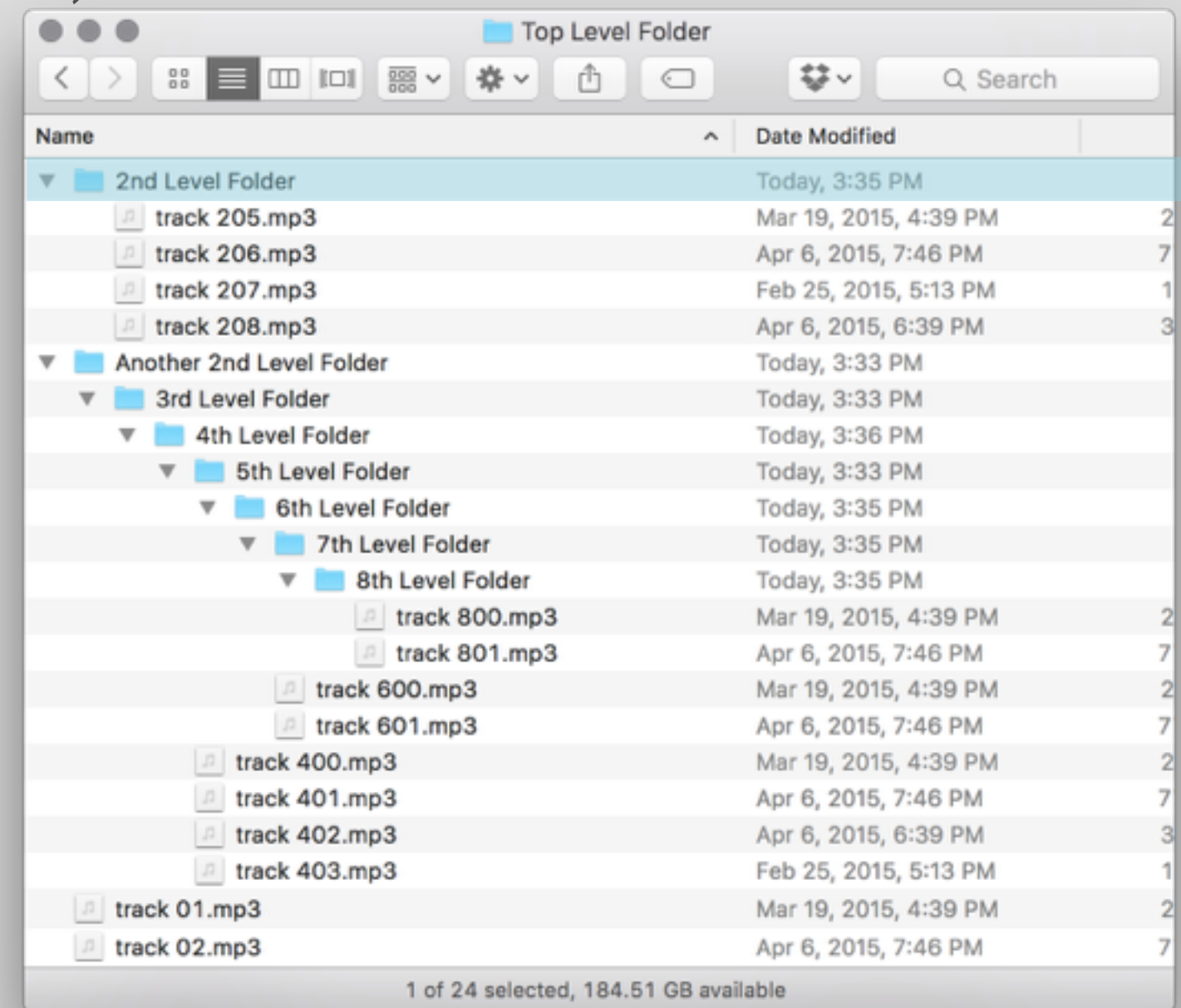
// call the function
listMP3Files("/Top Level Folder")
```



Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
        if (item is a folder)
            listMP3Files(name of item)
        end if
    end for
end function

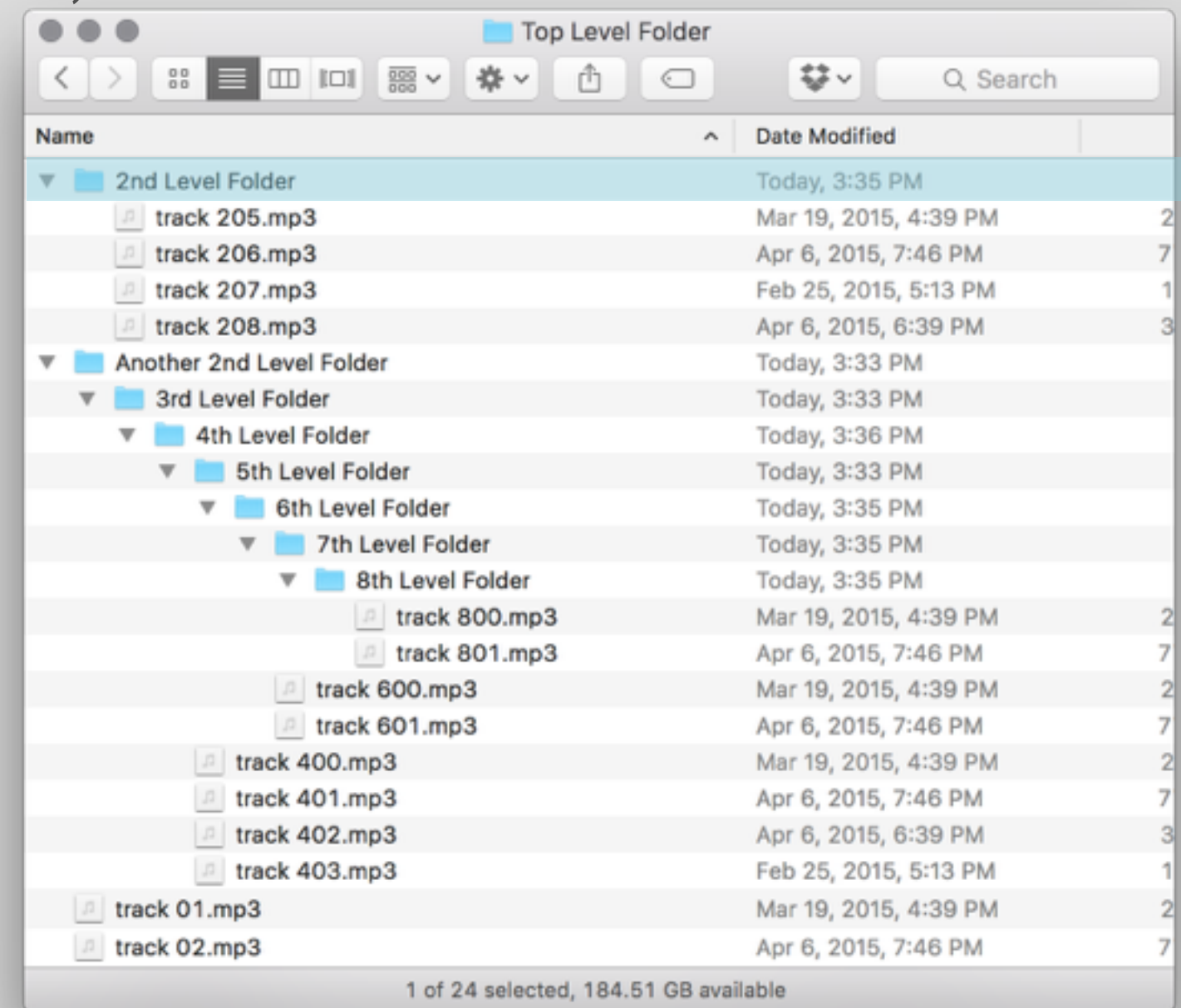
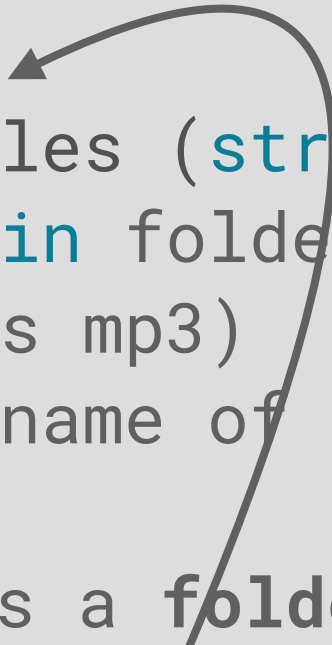
// call the function
listMP3Files("/Top Level Folder")
```



Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
  for each item in folder:currentFolder
    if (item is mp3)
      print(name of item)
    end if
    if (item is a folder)
      listMP3Files(name of item)
    end if
  end for
end function

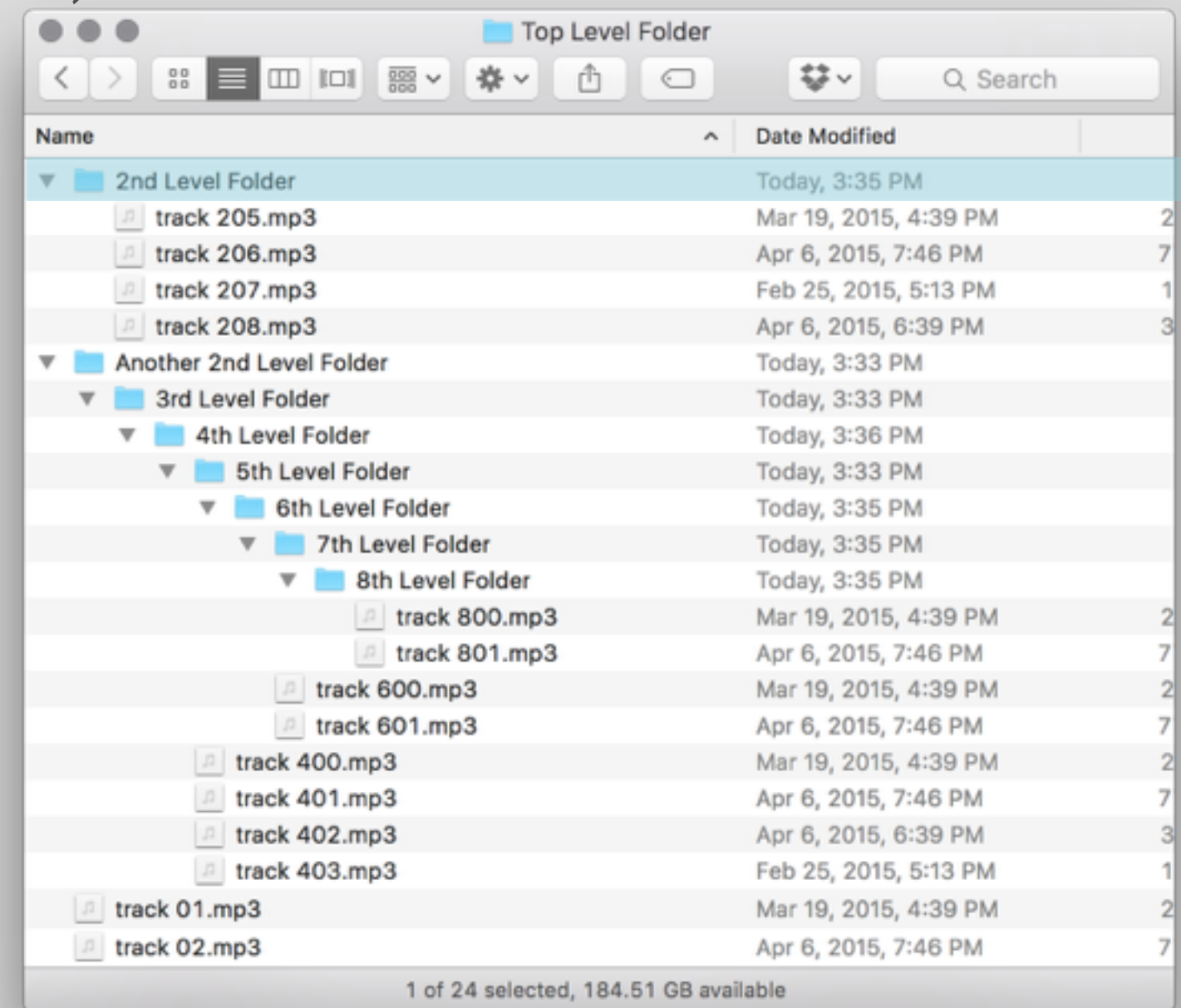
// call the function
listMP3Files("/Top Level Folder")
```



Recursive Function Calls

```
// pseudocode
function listMP3Files (string currentFolder)
    for each item in folder:currentFolder
        if (item is mp3)
            print(name of item)
        end if
        if (item is a folder)
            listMP3Files(name of item)
        end if
    end for
end function

// call the function
listMP3Files("/Top Level Folder")
```



Modular Code: Composite Data Types

Composite Data Types

Composite Data Types

```
string title = "Donkey Kong"
```

Composite Data Types

```
string title = "Donkey Kong"  
string publisher = "Nintendo"
```

Composite Data Types

```
string title = "Donkey Kong"  
string publisher = "Nintendo"  
int yearReleased = 1981
```


Composite Data Types

```
string title = "Donkey Kong"  
string publisher = "Nintendo"  
int yearReleased = 1981  
boolean completed = true
```

Composite Data Types

```
string title = "Donkey Kong"  
string publisher = "Nintendo"  
int yearReleased = 1981  
boolean completed = true  
  
string game2title = "Final Fantasy VII"
```

Composite Data Types

```
string title = "Donkey Kong"  
string publisher = "Nintendo"  
int yearReleased = 1981  
boolean completed = true  
  
string game2title = "Final Fantasy VII"  
string game2publisher = "Square"
```

Composite Data Types

```
string title = "Donkey Kong"  
string publisher = "Nintendo"  
int yearReleased = 1981  
boolean completed = true
```

```
string game2title = "Final Fantasy VII"  
string game2publisher = "Square"  
int game2yearReleased = 1997
```

Composite Data Types

```
string title = "Donkey Kong"  
string publisher = "Nintendo"  
int yearReleased = 1981  
boolean completed = true
```

```
string game2title = "Final Fantasy VII"  
string game2publisher = "Square"  
int game2yearReleased = 1997  
boolean game2completed = false
```

Defining a New Data Type

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}
```

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}
// create a variable of that type
```


Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}

// create a variable of that type
Game firstGame
```

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}

// create a variable of that type
Game firstGame

// use dot syntax
// to access member variables
```

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}

// create a variable of that type
Game firstGame
// use dot syntax
// to access member variables
firstGame.title = "Donkey Kong"
```

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}

// create a variable of that type
Game firstGame
// use dot syntax
// to access member variables
firstGame.title = "Donkey Kong"
firstGame.publisher = "Nintendo"
```

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}

// create a variable of that type
Game firstGame
// use dot syntax
// to access member variables
firstGame.title = "Donkey Kong"
firstGame.publisher = "Nintendo"
firstGame.yearReleased = 1981
```

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}

// create a variable of that type
Game firstGame
// use dot syntax
// to access member variables
firstGame.title = "Donkey Kong"
firstGame.publisher = "Nintendo"
firstGame.yearReleased = 1981
firstGame.completed = true
```

Defining a New Data Type

```
// define the new type
struct Game {
    string title
    string publisher
    int yearReleased
    boolean completed
}

// create a variable of that type
Game firstGame
// use dot syntax
// to access member variables
firstGame.title = "Donkey Kong"
firstGame.publisher = "Nintendo"
firstGame.yearReleased = 1981
firstGame.completed = true
```

firstGame

title	publisher	yearReleased	completed
"Donkey Kong"	"Nintendo"	1981	true

Defining a New Data Type

```
// define the new type
```

```
struct Game {  
    string title  
    string publisher  
    int yearReleased  
    boolean completed  
}
```

```
// create a variable of that type
```

```
Game firstGame
```

```
// use dot syntax
```

```
// to access member variables
```

```
firstGame.title = "Donkey Kong"
```

```
firstGame.publisher = "Nintendo"
```

```
firstGame.yearReleased = 1981
```

```
firstGame.completed = true
```

firstGame

title	publisher	yearReleased	completed
"Donkey Kong"	"Nintendo"	1981	true

```
// create another
```

```
Game secondGame
```

```
secondGame.title = "Final Fantasy VII"
```

```
secondGame.publisher = "Square"
```

```
secondGame.yearReleased = 1997
```

```
secondGame.completed = false
```


Defining a New Data Type

```
// define the new type
```

```
struct Game {  
    string title  
    string publisher  
    int yearReleased  
    boolean completed  
}
```

```
// create a variable of that type
```

```
Game firstGame
```

```
// use dot syntax
```

```
// to access member variables
```

```
firstGame.title = "Donkey Kong"
```

```
firstGame.publisher = "Nintendo"
```

```
firstGame.yearReleased = 1981
```

```
firstGame.completed = true
```

firstGame

title	publisher	yearReleased	completed
"Donkey Kong"	"Nintendo"	1981	true

secondGame

title	publisher	yearReleased	completed
"Final Fantasy VII"	"Square"	1997	false

```
// create another
```

```
Game secondGame
```

```
secondGame.title = "Final Fantasy VII"
```

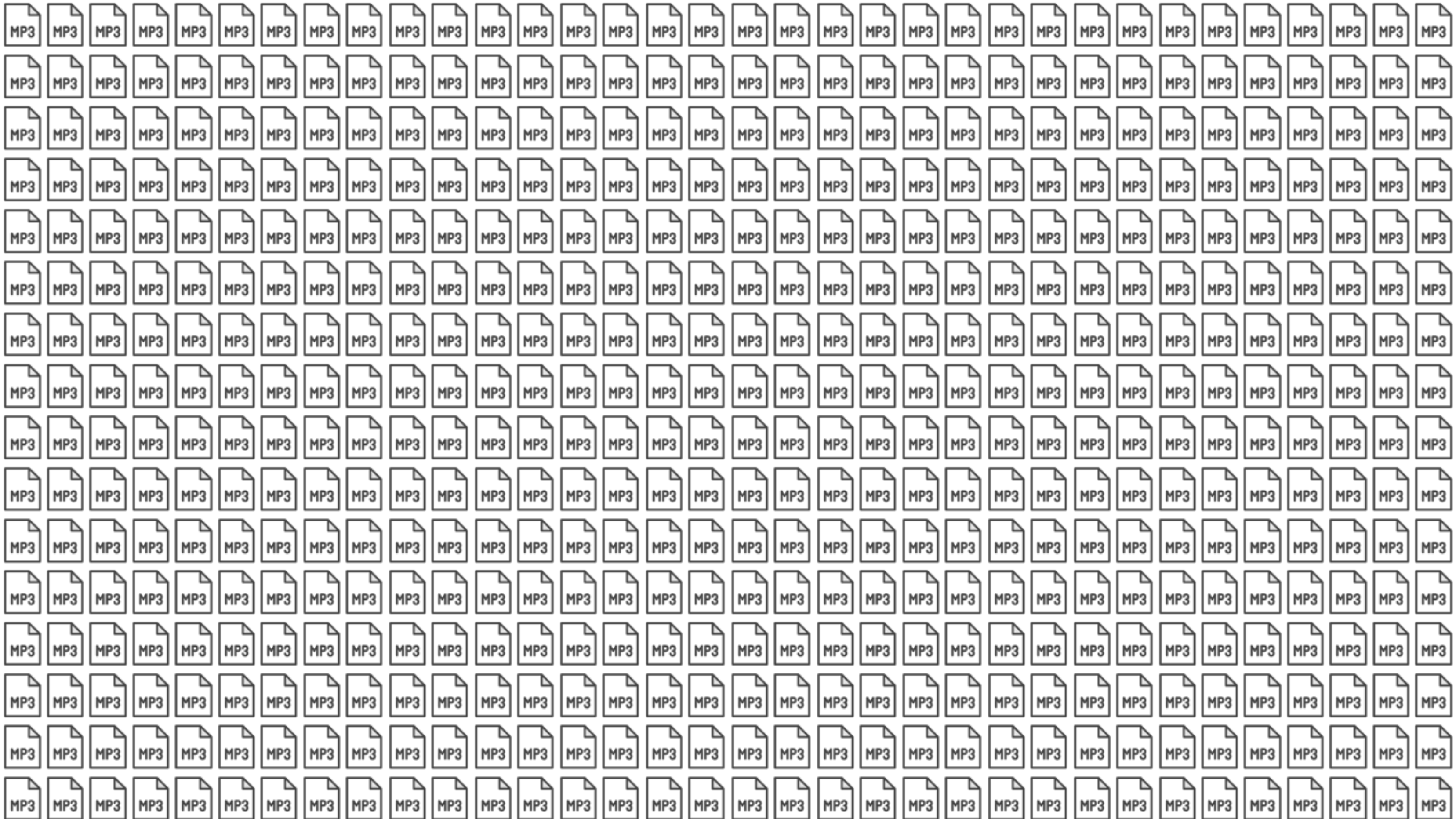
```
secondGame.publisher = "Square"
```

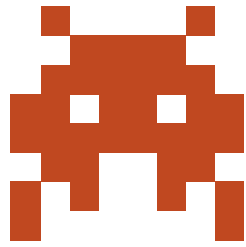
```
secondGame.yearReleased = 1997
```

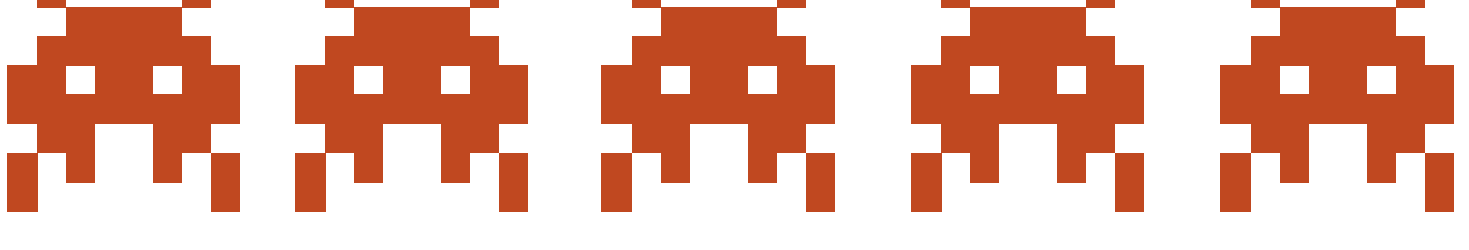
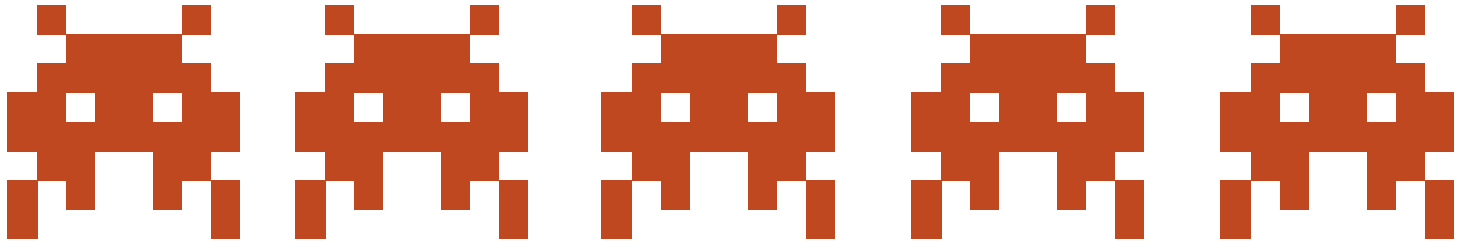
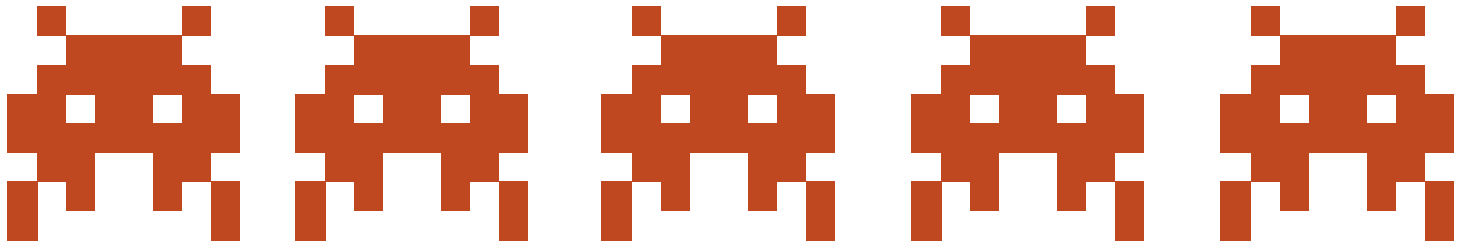
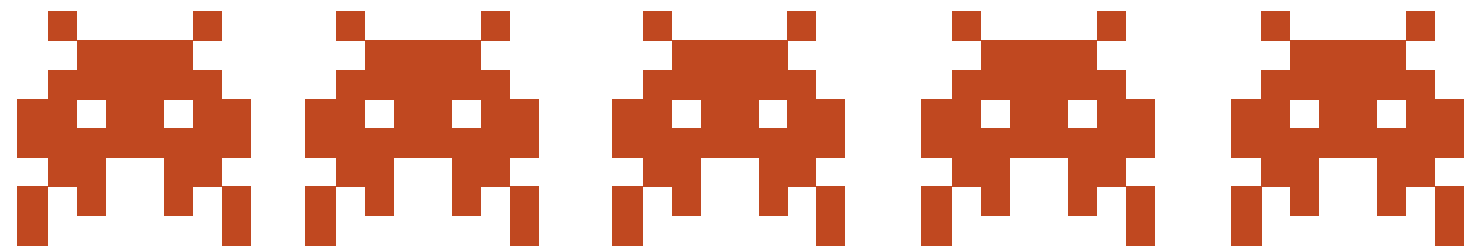
```
secondGame.completed = false
```

Using Arrays and Collections









Arrays

Arrays

1

myInt

Arrays

1

myInt

1	3	5	7	9
----------	----------	----------	----------	----------

myArray

Declaring an Array of 5 Integers

C `int myArray[5] = {1,3,5,7,9};`

Swift `var myArray = [1,3,5,7,9]`

Ruby `myArray = [1,3,5,7,9]`

JavaScript `var myArray = [1,3,5,7,9]`

C# `int[] myArray = {1,3,5,7,9};`

Declaring an Array of 5 Integers

C `int myArray[5] = {1, 3, 5, 7, 9};`

Swift `var myArray = [1, 3, 5, 7, 9]`

Ruby `myArray = [1, 3, 5, 7, 9]`

JavaScript `var myArray = [1, 3, 5, 7, 9]`

C# `int[] myArray = {1, 3, 5, 7, 9};`

Declaring an Array of 5 Integers

C `int myArray[5] = {1,3,5,7,9};`

Swift `var myArray = [1,3,5,7,9]`

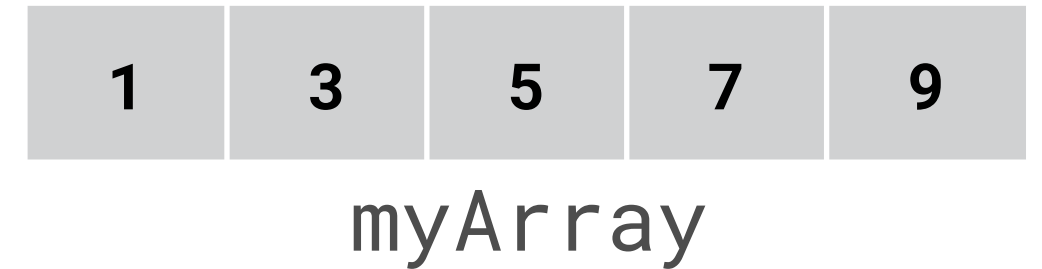
Ruby `myArray = [1,3,5,7,9]`

JavaScript `var myArray = [1,3,5,7,9]`

C# `int[] myArray = {1,3,5,7,9};`

Declaring an Array of 5 Integers

C `int myArray[5] = {1, 3, 5, 7, 9};`



Swift `var myArray = [1, 3, 5, 7, 9]`

Ruby `myArray = [1, 3, 5, 7, 9]`

JavaScript `var myArray = [1, 3, 5, 7, 9]`

C# `int[] myArray = {1, 3, 5, 7, 9};`

Array Elements

1	3	5	7	9
----------	----------	----------	----------	----------

myArray

Array Elements

	1	3	5	7	9
index:	[0]	[1]	[2]	[3]	[4]
	myArray				

Array Elements

	1	3	5	7	9
index:	[0]	[1]	[2]	[3]	[4]

myArray

```
// to access the middle element  
print( myArray[2] )
```


Array Elements

	1	3	5	7	9
index:	[0]	[1]	[2]	[3]	[4]

myArray

```
// to access the middle element  
print( myArray[2] )
```

"5"

Array Elements

	1	3	5	7	9
index:	[0]	[1]	[2]	[3]	[4]

myArray

```
// to access the middle element  
print( myArray[2] )
```

"5"

```
// to change the last element  
myArray[4] = 99
```

Array Elements

	1	3	5	7	99
index:	[0]	[1]	[2]	[3]	[4]

myArray

```
// to access the middle element  
print( myArray[2] )
```

"5"

```
// to change the last element  
myArray[4] = 99
```

Array Elements

	1	3	5	7	99
index:	[0]	[1]	[2]	[3]	[4]
myArray					

```
// to access the middle element  
print( myArray[2] )
```

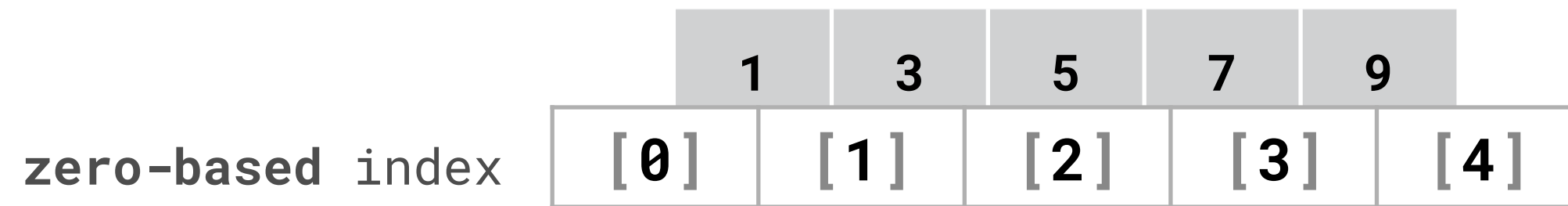
 "5"

```
// to change the last element  
myArray[4] = 99
```

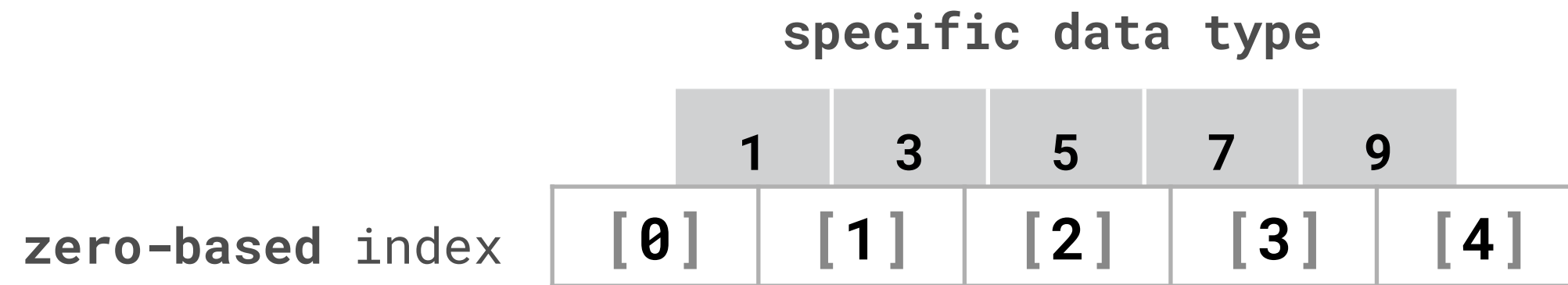
Basic Array Characteristics

1	3	5	7	9
[0]	[1]	[2]	[3]	[4]

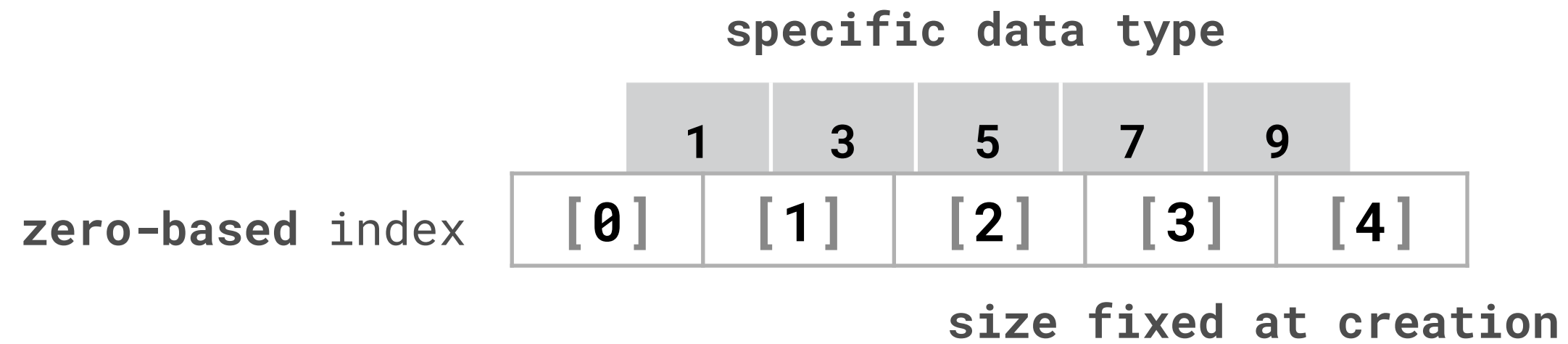
Basic Array Characteristics



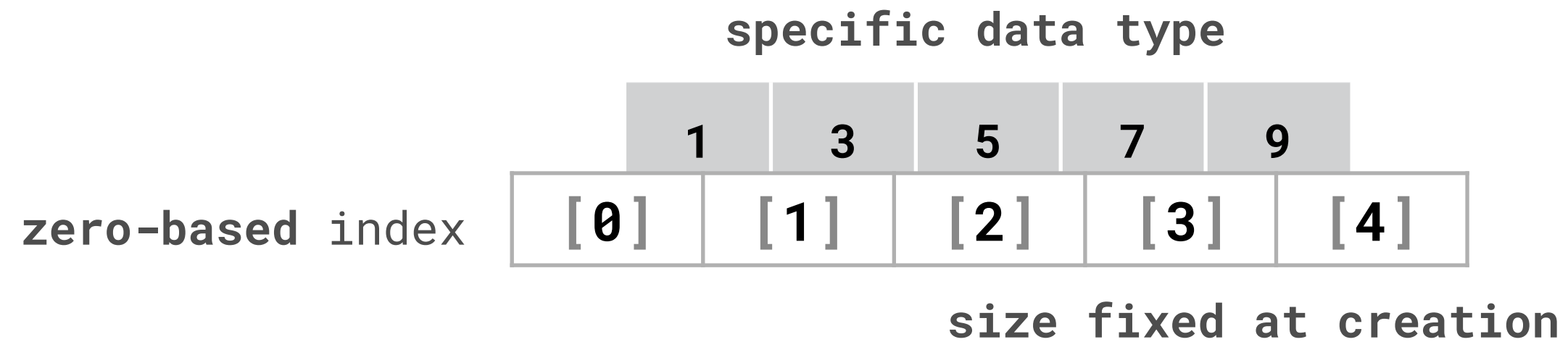
Basic Array Characteristics



Basic Array Characteristics



Basic Array Characteristics



```
// array creation can use a variable  
int myArray[someVariable];
```

Collections

Collections

Basic Arrays

Fixed type, immutable,
zero-based

Collections

Basic Arrays

Fixed type, immutable,
zero-based

Dynamic (Mutable) Arrays

Can add and remove
elements

Collections

Basic Arrays

Fixed type, immutable,
zero-based

Dynamic (Mutable) Arrays

Can add and remove
elements

Type-flexible Arrays

Integers & strings &
objects, oh my!

Collections

Basic Arrays

Fixed type, immutable,
zero-based

Dynamic (Mutable) Arrays

Can add and remove
elements

Type-flexible Arrays

Integers & strings &
objects, oh my!

Associative Arrays

Choose your own index

New York	Arizona	Utah	Missouri	California
[NY]	[AZ]	[UT]	[MO]	[CA]

Collections

Basic Arrays

Fixed type, immutable,
zero-based

Dynamic (Mutable) Arrays

Can add and remove
elements

Type-flexible Arrays

Integers & strings &
objects, oh my!

Associative Arrays

Choose your own index

New York	Arizona	Utah	Missouri	California
[NY]	[AZ]	[UT]	[MO]	[CA]

Collections

Basic Arrays

Fixed type, immutable,
zero-based

Dynamic (Mutable) Arrays

Can add and remove
elements

Type-flexible Arrays

Integers & strings &
objects, oh my!

Associative Arrays

Choose your own index

New York	Arizona	Utah	Missouri	California
[NY]	[AZ]	[UT]	[MO]	[CA]

Collections

Basic Arrays

Fixed type, immutable,
zero-based

Dynamic (Mutable) Arrays

Can add and remove
elements

Type-flexible Arrays

Integers & strings &
objects, oh my!

Associative Arrays

Choose your own index

Others

Dictionaries, Queues, Lists,
Stacks, Hash Tables, (etc.)

New York	Arizona	Utah	Missouri	California
[NY]	[AZ]	[UT]	[MO]	[CA]

Introducing Object-Oriented Programming

Languages with OO Features

C

C++

Swift

Objective-C

Python

Ruby

Java

R

VB.NET

C#

Perl

PHP

JavaScript

Delphi

Groovy

Boo

Lua

Scala

D

F#

Languages with OO Features

C

C++

Swift

Objective-C

Python

Ruby

Java

R

VB.NET

C#

Perl

PHP

JavaScript

Delphi

Groovy

Boo

Lua

Scala

D

F#

The Big Idea

The Big Idea



```
// variables (data)
```

```
birthDate
```

```
memberType
```

```
firstName
```

```
lastName
```

```
email
```

```
itemsInCart
```

```
orderTotal
```

```
orderDate
```

```
discountCode
```

```
// etc.
```

```
// functions (behavior)
```

```
function changeEmail { ... }
```

```
function addItem { ... }
```

```
function changeAddress { ... }
```

```
function calculateShipping { ... }
```

```
function calculateTotal { ... }
```

```
function sendNewsletter { ... }
```

```
function processPayment { ... }
```

```
function referFriend { ... }
```

```
function sendSample { ... }
```

```
// etc.
```

Understanding Objects

Begin with non-technical language

"A customer can begin a new order
by adding an item to the shopping cart"

Understanding Objects

Begin with non-technical language

"A **customer** can begin a new **order**
by adding an **item** to the **shopping cart**"

Understanding Objects

Begin with non-technical language

"A **customer** can begin a new **order**
by adding an **item** to the **shopping cart**"

Potential objects: **customer, order, item, shopping cart...**

Understanding Objects

Begin with non-technical language

"A **customer** can begin a new **order**
by adding an **item** to the **shopping cart**"

Potential objects: **customer, order, item, shopping cart...**

For a media player: **album, track, playlist...**

Understanding Objects

Begin with non-technical language

"A **customer** can begin a new **order**
by adding an **item** to the **shopping cart**"

Potential objects: **customer, order, item, shopping cart...**

For a media player: **album, track, playlist...**

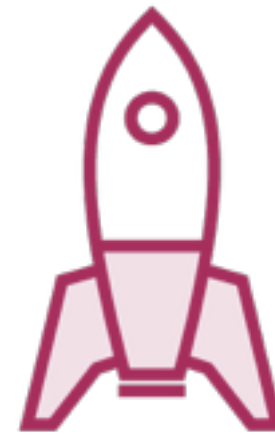
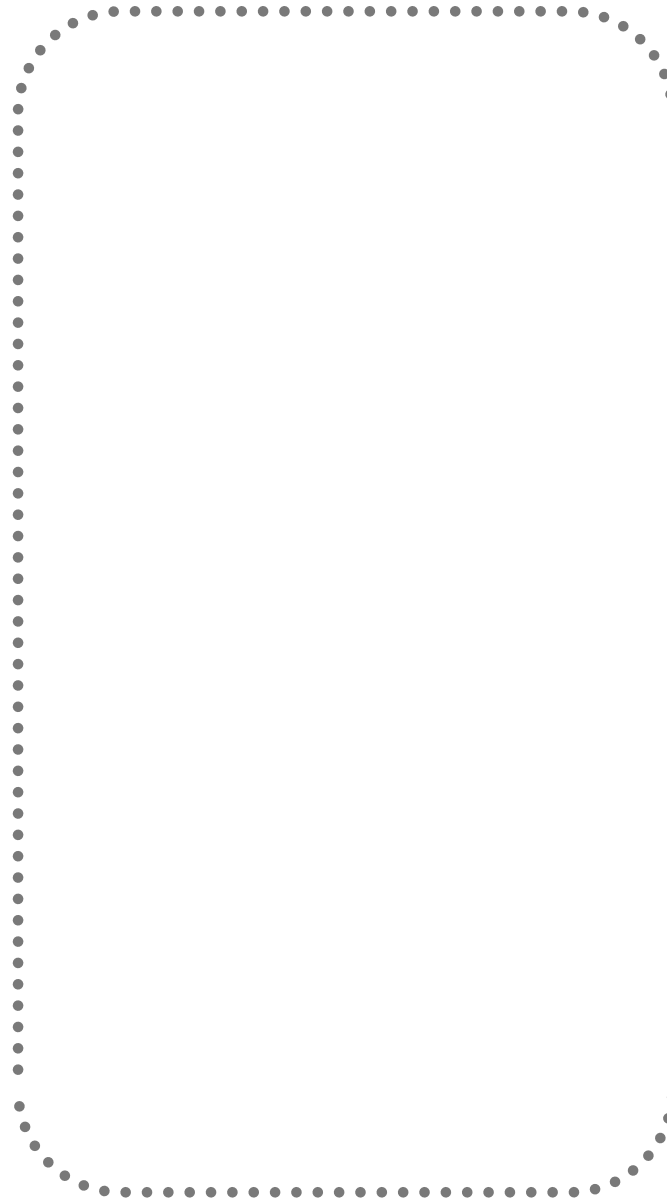
For a game: **spaceship, enemy, asteroid, missile...**

Understanding Objects

Begin with non-technical language

Objects Contain **Data and Functionality**

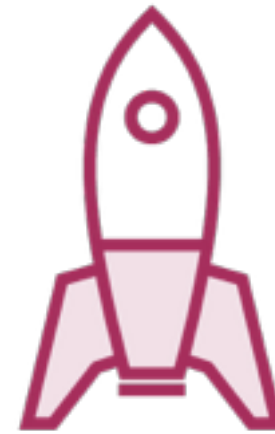
Spaceship



Objects Contain **Data and Functionality**

Spaceship

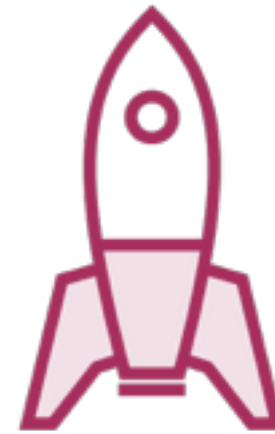
```
// data  
positionX  
positionY  
shieldLevel  
color  
name
```



Objects Contain **Data and Functionality**

Spaceship

```
// data  
positionX  
positionY  
shieldLevel  
color  
name  
  
// behavior  
fly()  
fireMissile()  
explode()  
...
```



OO Terminology



OO Terminology

Class

Object

OO Terminology

Class

Object

How we define an object

OO Terminology

Class

How we define an object

Object

The object itself

OO Terminology

Class

How we define an object

"The Blueprint"

Object

The object itself

OO Terminology

Class

How we define an object

"The Blueprint"

Object

The object itself

"The house made from the blueprint"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

Object

The object itself

"The house made from the blueprint"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

One class can be used to
make multiple objects

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

One class can be used to
make multiple objects

The class comes first!

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

One class can be used to
make multiple objects

The class comes first!

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

One class can be used to
make multiple objects

The class comes first!

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

One class can be used to
make multiple objects

The class comes first!

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

One class can be used to
make multiple objects

The class comes first!

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

OO Terminology

Class

How we define an object

"The Blueprint"

"The Recipe"

One class can be used to
make multiple objects

The class comes first!

Object

The object itself

"The house made from the blueprint"

"The dish made from the recipe"

Defining a Class / Creating an Object

Defining a Class / Creating an Object

```
class Spaceship {
```

```
}
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data  
    int positionX  
    int positionX  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()          {...}  
}
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionX  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()           {...}  
  
}
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionX  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()           {...}  
  
}
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionX  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()           {...}  
  
}
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionX  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()           {...}  
}
```

instantiation

```
// create a new object from that class  
var mainSpaceship = new Spaceship()
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionY  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()          {...}  
}
```

instantiation

```
// create a new object from that class  
var mainSpaceship = new Spaceship()  
// access properties
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionX  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()           {...}  
}
```

instantiation

```
// create a new object from that class  
var mainSpaceship = new Spaceship()  
// access properties  
mainSpaceship.name = "Nostromo"
```


Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionY  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()           {...}  
}
```

instantiation

```
// create a new object from that class  
var mainSpaceship = new Spaceship()  
// access properties  
mainSpaceship.name = "Nostromo"  
// call methods
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionY  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()          {...}  
}
```

instantiation

```
// create a new object from that class  
var mainSpaceship = new Spaceship()  
// access properties  
mainSpaceship.name = "Nostromo"  
// call methods  
mainSpaceship.fireMissile()
```

Defining a Class / Creating an Object

```
class Spaceship {  
    // data      properties  
    int positionX  
    int positionY  
    int shieldLevel  
    string color  
    string name  
    // etc.  
  
    methods  
    // behavior  
    func fireMissile() {...}  
    func explode()      {...}  
    func changeColor() {...}  
    func fly()          {...}  
}
```

instantiation

```
// create a new object from that class  
var mainSpaceship = new Spaceship()  
// access properties  
mainSpaceship.name = "Nostromo"  
// call methods  
mainSpaceship.fireMissile()  
  
// instantiate another object  
var secondSpaceship = new Spaceship()  
// access properties  
mainSpaceship.name = "Heart of Gold"  
// call methods  
mainSpaceship.explode()
```