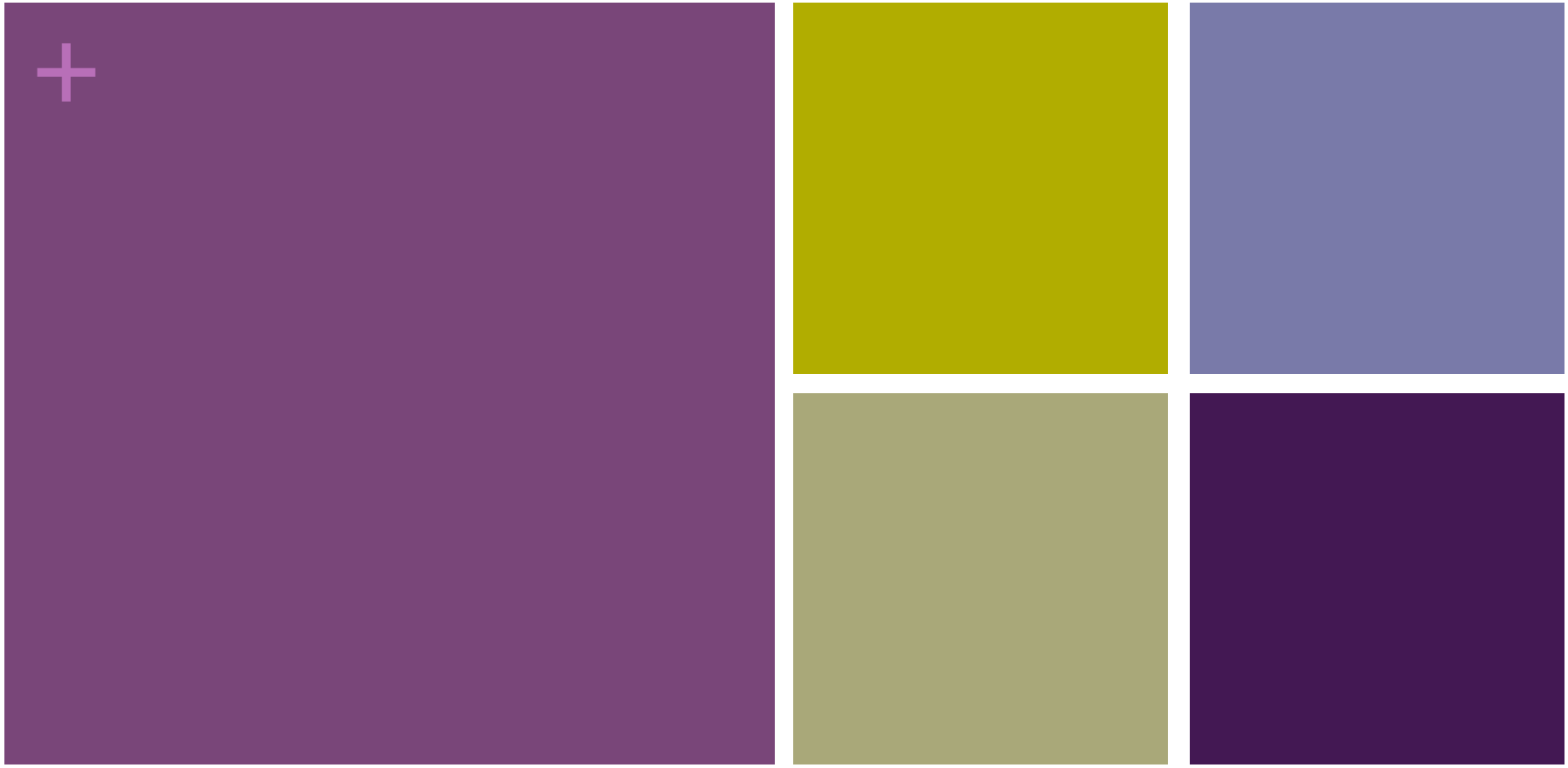


Elementary Java Programming

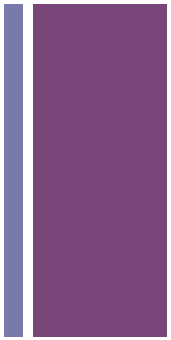
Introduction to Computer Science CSCI- UA.0101
Lecture 2

Agenda Day 2

- Anatomy of a Java program
- Variables and data types
- Getting user input
- Intro to Dialog Boxes
- Strings

Anatomy of a Java program

Anatomy of a Simple Java Program



Welcome.java

```
public class Welcome
{
      public static void main(String[] args)
      {
            // This will print "Hello World"
            System.out.println("Hello, World!");
      }
}
```

Anatomy of a Simple Java Program: class definition

Welcome.java

```
public class Welcome
{
     public static void main(String[] args)
     {
          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

- Every Java program must have at least one class
- Each class needs to be given a unique name
- By convention, class names start with an uppercase letter
- The “public” modifier indicates that this class is fully accessible to other classes (we will talk more about this later in the term)

Anatomy of a Simple Java Program: blocks

Welcome.java

```
public class Welcome
{
     public static void main(String[] args)
     {
          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

- Unlike Python, Java requires you to define “blocks” that delimit the beginning and end of a region of source code
- This is done using the “curly” brace. Curly braces come in pairs – the open brace denotes the beginning of a block and the closing brace denotes the end of a block
- Your classes must be fully contained with a “class block”
- Blocks can be nested inside of one another, which we will see in a moment when we examine the main method

Anatomy of a Simple Java Program: block styles

Next Line Style

Welcome.java

```
public class Welcome
{
     public static void main(String[] args)
     {
          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

End of Line Style

Welcome.java

```
public class Welcome {

     public static void main(String[] args){

          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

Anatomy of a Simple Java Program: main method

Welcome.java

```
public class Welcome
{
     public static void main(String[] args)
     {
          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

- A method is a Java construct that can contain programming statements
- Methods can be thought of as a "function" in Python
- All methods must be followed by a "method block" (pair of open and closed curly braces)
- The "main" method is a special method that Java will execute before any other methods – it is the "entry point" to your program

Anatomy of a Simple Java Program: main method (cont.)

Welcome.java

```
public class Welcome
{
     public static void main(String[] args)
     {
          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

- In this case our main method uses the following syntax:
  - **public**: denotes that the method is accessible outside the class
  - **static**: we will discuss this when we get to object oriented programming
  - **void**: this is the return type of the method. The main method returns nothing, so we say it has a "void" return type.
  - **(String[] args)**: the arguments that the main method expects to receive when it starts up. In this case it expects to receive an array (list) of Strings

Anatomy of a Simple Java Program: comments

Welcome.java

```
public class Welcome
{
     public static void main(String[] args)
     {
          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

- Java supports two different commenting styles – single line and multi line
- A single line comment is denoted with a double slash (i.e. "//")
- A multiline comment begins with a "/*" character sequence and ends with the "*/" character sequence

Anatomy of a Simple Java Program: statements

Welcome.java

```
public class Welcome
{
     public static void main(String[] args)
     {
          // This will print "Hello World"
          System.out.println("Hello, World!");
     }
}
```

- "System.out.println" is a statement which can be used to display information on the console
- Statements in Java must end with the semicolon character
- "System.out.println" will print out text and terminate it with a line break. To print out text without a line break use the "print" method:

  System.out.print("text")

Writing a Java Program (command line)

- Write your source code (in any text editor) and save it with a java extension. For example **`HelloWorld.java`**

- Compile your source code into byte code using the Java compiler:

```
javac HelloWorld.java
```

- Execute your byte code through the Java Virtual Machine (JVM):

```
java HelloWorld
```

Basic Java Program Structure

```
/*
 *       COMMENTS: AUTHOR, PROBLEM
 */

public class ProgramStructure
{
        public static void main(String[] args)
        {
                //DECLARATIONS


                //INPUT



                //PROCESSING



                //OUTPUT


        }
}
```

+

Variables & Data Types

Variables

- A variable is a storage location and an associated symbolic identifier which contains some known or unknown quantity of information known as a value.
- Variables allow you to temporarily store information during the execution of your programs.
- We call them “variables” because they can “vary” over the life of your program – values stored within a variable can be updated as necessary

Java is “Strictly Typed”

- Java is a strictly typed language, meaning that you must pre- declare the type of data you wish to store within a variable before the variable can be created
- This allows the Java virtual machine to know how much memory needs to be allocated in order for your program to run efficiently
- Once a variable has been declared to be a specific type it cannot hold a value of another type. This means that if you create a variable to hold an integer it can not holder a floating point value later on in your program – it can only be used to hold an integer.

Numeric Data Types: Integers

Name	Storage Size	Range
byte	1 byte (8 bit signed)	-128 to 127
short	2 bytes (16 bit signed)	-32,768 to 32,767
int	4 bytes (32 bit signed)	-2,147,483,648 to 2,147,483,647
long	8 bytes (64 bit signed)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808

Numeric Data Types: Integers

Name	Storage Size	Range
byte	1 byte (8 bit signed)	-128 to 127
short	2 bytes (16 bit signed)	-32,768 to 32,767
int	4 bytes (32 bit signed)	-2,147,483,648 to 2,147,483,647
long	8 bytes (64 bit signed)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808

Numeric Data Types: Floating Point

Name	Storage Size	Range
float	4 bytes (32 bit IEEE 754)	-3.4028235 E+38 to 3.4028235 E+38
double	8 bytes (64 bit IEEE 754)	-1.7976931348623157 E+308 to 1.7976931348623157 E+308

Numeric Data Types: Floating Point

Name	Storage Size	Range
float	4 bytes (32 bit IEEE 754)	-3.4028235 E+38 to 3.4028235 E+38
double	8 bytes (64 bit IEEE 754)	-1.7976931348623157 E+308 to 1.7976931348623157 E+308

Declaring a variable

- You can declare a variable to be used in your program using the following syntax:

```
data_type identifier;
```

- For example:

```
int myIntegerVariable;
short myShortVariable;
long myLongVariable;
```

- Example: ***Declarations.java***

Declaring multiple variables

- You can also declare multiple variables at the same time. For example:

```
// declare 3 integers
int a, b, c;
```

The Assignment Operator

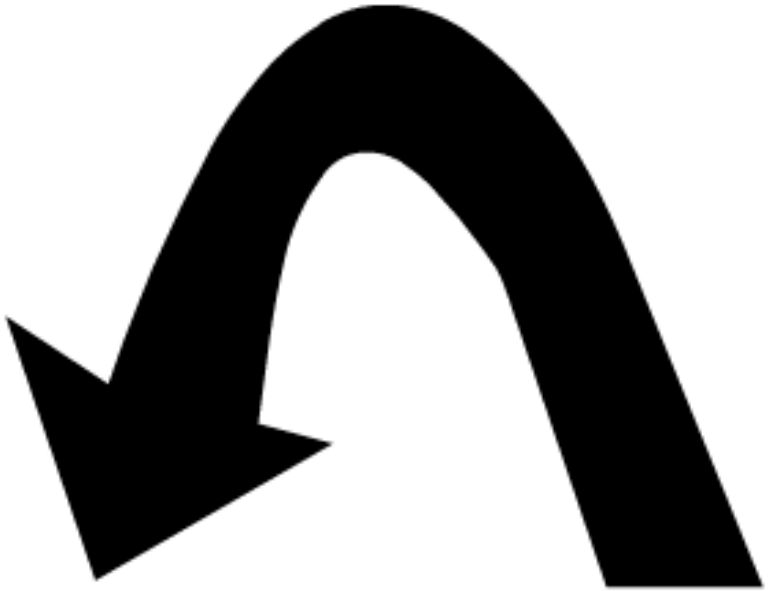
- In Java, the assignment operator is the single equals sign (the same as in Python)
- You can assign a value to a variable by using the following syntax:

```
int myInteger = 100;
```

- You do not necessarily need to assign data to a variable when you declare it. For example, the following statement is valid:

```
int myInteger;
myInteger = 100;
```

The Assignment Operator



```
int myInteger = 1000;
```

Printing variables to the console

- You can use a variable identifier inside of a call to the System.out.println() method. Example:

```
int a = 100;
System.out.println(a);
```

- You can print multiple items to the console using the concatenation operator (+). For example:

```
int a = 100;
int b = 100;

System.out.println("a: " + a + " and b: " + b);
```

Identifiers

- An “identifier” is any name that you define inside of your program. These include:
  - Class names
  - Variable names
  - Method names
- Identifiers must conform to the following rules:
  - Must contain only alphabetic, numeric,“_” or “$” characters
  - Cannot begin with a numeric character
  - Cannot be a Java reserved word (i.e. you can’t create a class called “int”)
  - Can be of any length

Identifier Naming Conventions

- Try and use lowercase names for variables and methods. For example:

```
int days;
double salary;
```

- If your name contains multiple words (i.e. a compound word) then you should try and capitalize the letter of subsequent words. Example:

```
int daysOfWeek = 7;
double yearlySalary = 50000.0;
```

Numeric Operations

- Java contains a number of built in numeric operators that can be used to perform arithmetic operations.

Numeric Operations

Name	Meaning	Example	Result
+	Addition	10 + 2	12
-	Subtraction	100.0 - 0.5	99.5
*	Multiplication	100 * 2	200
/	Division	1.0 / 2.0	0.5
%	Remainder (mod)	20 % 3	2

Order of Operations

- Java evaluates expressions using standard arithmetic order of operations
  1. Parenthesis
  2. Multiplication, Division & Remainder Operation, read left to right
  3. Addition, Subtraction, read left to right

Mixed Type Expressions

- Java allows you to freely mix numeric types when constructing an expression.
- When all operands are integer data types, the result will be an integer:

```
System.out.println( 5 + 2 ); //  7
```

- When one operand is a floating point type, the result will be a floating point number:

```
System.out.println( 5 + 2.0 ); //  7.0
```

Mixed Type Expressions

- Unlike Python, Java does not have a special integer division operator (//)
- To perform integer division you must ensure that all operands are integers. For example:

```
System.out.println (5 / 2);   // 2
```

- To perform normal mathematical division you need to make sure at that at least one operand is a floating point number:

```
System.out.println (5 / 2.0); //  2.5
```

Named Constants

- The value stored in a variable can change during the execution of a program
- However, there are times in which you want to prevent a value from being changed.
- You can declare a variable to be “final” in Java – this prevents any aspect of your program from changing its value. We call these “named constants”
- It’s customary to capitalize all letters of a named constant.
  Example:

```
final double PI = 3.14159;
```

Getting User Input

Getting User Input

- Just like in Python, you can have Java prompt the user and accept input from the console. This can be accomplished using an instance of the Scanner class.
- The Scanner class is a class that comes bundled in the Java library, and it exists in the java.util package, which means you need to import it before you can use it in your program.

The Java Library

- Java comes pre-installed with a huge library of pre-written classes that you can use in your programs.
- The full library – along with documentation – is available on Oracle's website:
  - http://docs.oracle.com/javase/7/docs/api/
- Most Java programmers are fluent with a core set of classes in the Java library.
- It's more important to be able to know how to work with the library rather than to know how to use every single class – you can always look up a particular class when you encounter the need to use it!

Importing a class from the library

- Before you can use a class in the Java library you must first import it into your program
- You can do this by using the import statement
- The import statement is placed above your class definition and looks like the following.

```
import package.className
```

The Scanner Class

- The Scanner class is not a "static" class like the System class. This means that you can't just use it. You first need to make an "instance" of it in your program. Here's how to get started:

```
import java.util.Scanner;

public class Welcome {
      public static void main (String args[])  {
            Scanner input = new  Scanner(System.in);
      }
}
```

- The "new" keyword tells Java that we want to make a new Scanner object. We usually refer to this as an "instance" or as an "object"

Reading Numeric Data

- Once you have created an instance of the Scanner class you can use it to read input from the console using some of its pre-created methods.
- Example:

```
Scanner input = new Scanner(System.in);

System.out.print("Give me an int: ");
int myInt = input.nextInt();

System.out.println("Give me a double: ");
double myDouble = input.nextDouble();

System.out.println("You gave me: " + myInt + " & " + myDouble);
```

Reading Numeric Data

Scanner Method	Resulting Data Type
`nextByte()`	Reads a byte integer Reads
`nextShort()`	a short integer Reads an int
`nextInt()`	integer Reads a long
`nextLong()`	integer Reads a float
`nextFloat()`	number Reads a double
`nextDouble()`	number

Reading String Data

- You can use the nextLine() method to read in a String from the console
- The nextLine() method will continue to read data until it encounters a line break (i.e. the user hit the Enter key)
- Example:

```
Scanner input = new Scanner(System.in);

System.out.print("What's your name? ");
String name = input.nextLine();

System.out.println("Welcome, " + name);
```

Warning!

- It is not recommended that you call any of the numeric input methods (nextInt(), nextDouble(), etc) after you call the nextLine() method
- When you call a numeric input method you are only reading the numeric input the user typed in – there is still a line break sitting on the input buffer.
- If you call nextLine() after nextInt() you will read in the line break that is already on the buffer and will ignore any new input.

Warning!

This won't work …

```
System.out.print("int: ");
int myInt = input.nextInt();

System.out.print("int2 " );
int myInt2 = input.nextInt();

System.out.print("String: ");
String myString = input.nextLine();

System.out.println("I got .... ");
System.out.println(myInt + "," + myInt2 + ","
+ myString);
```

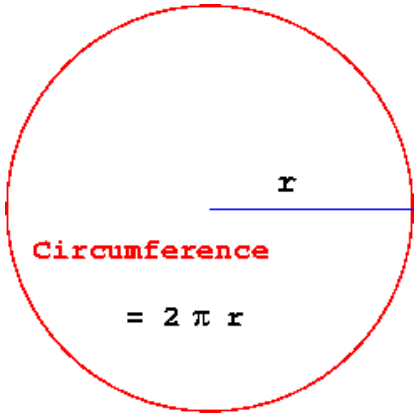
Solution: Use two Scanners

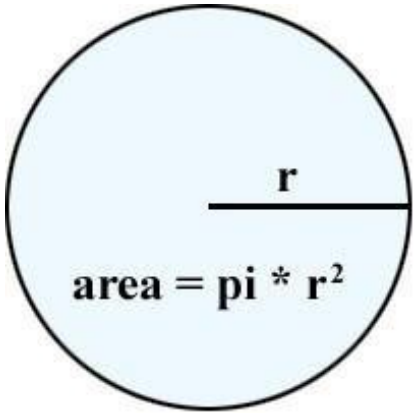
- You can instantiate two Scanner objects and use one to read in numeric values and one to read in String values

```
// create two scanners
Scanner numberInput = new Scanner(System.in);
Scanner stringInput = new  Scanner(System.in);

// read in our integers using the numberScanner
System.out.print("int:  ");
int myInt =  numberInput.nextInt();

System.out.print("int2: "  );
int myInt2 =  numberInput.nextInt();

// read in our Strings using our stringScanner
System.out.print("String:  ”);
String myString =  stringInput.nextLine();

System.out.println("I got .... ");
System.out.println(myInt + "," + myInt2 + ","  +  myString);
```

Programming Challenge

- Write a program that asks the user to enter a radius as a double
- Calculate the area and circumference of a circle with that radius
- Define PI as a constant double of 3.14159
- Example: ***Circle.java***





Programming Challenge

- Write a program that reads the following information from the user and then generates a payroll statement.

  Employee name
  Number of hours worked Hourly pay rate
  Fed tax withholding rate

- Example:

```
Employee Name: Craig
Hours worked: 40
Hourly rate: 10
Income tax rate (i.e. 15 for 15%): 20

Employee: Craig
Pay before taxes: 400.0
Fed withholding: 80.0
Net pay: 320.0
```

- Example: ***PayrollConsole.java***

Intro to Dialog Boxes

Displaying Text in a Message Dialog Box

- Java has an extensive library for constructing Graphical User Interfaces (GUIs)
- This library, called "swing", can be imported into your program in order to give you access to Java's GUI functionality
- The "swing" library is housed in the following "package" (note that some packages are nested):

  `javax.swing`

- The "swing" library contains a number of classes – let's check them out:

  http://docs.oracle.com/javase/7/docs/api/

Displaying Text in a Message Dialog Box

- The JOptionPane class in the “swing” library has the ability to create a pop-up window that can be used to display strings of text to the user
- To use the JOptionPanel class you must first import it into your class as follows:

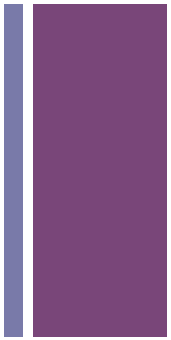
```
import javax.swing.JOptionPane;

public class Welcome
{
     // class code to go here
}
```

Displaying Text in a Message Dialog Box

- We can then use the JOptionPane to display text by calling the "showMessageDialog" method inside the JOptionPanel class.
- Methods on other classes can be called using "dot" syntax. To call the "showMessageDialog" method on the JOptionPanel class you can do the following:

```
JOptionPane.showMessageDialog(null, "Hello!");
```

Displaying Text in a Message Dialog Box



Welcome.java

```
import javax.swing.JOptionPane;

public class Welcome
{
     public static void main(String[] args)
     {
          // display a GUI pop-up box with some text
          JOptionPane.showMessageDialog(null, "Hello, World!");
     }
}
```

More about the import statement

- The import statement is designed to make it easy to utilize classes outside of your program
- There are two types of import statements
  - Specific import statements: You specify the exact class you wish to import into your program. Example:

```
import javax.swing.JOptionPane;
```

  - Wildcard import statements: You specify that you want to import ALL classes inside a specific package into your program. Example:

```
import javax.swing.*;
```

So what about System.out.println()?

- As you remember from a few slides ago, System.out.println() lets you print text to the console
- However, you never imported a package for this method call, so how does Java know what to do when you call it?
- If you look at the Java library you will see that the System class is organized in the “java.lang” package. This package is automatically imported into all Java classes for you.

Programming Challenge

- Write a program that converts a temperature in Fahrenheit into its Celsius equivalent.
- Use the formula on the right to perform your calculation

$$\text{Celsius} = \frac{5}{9}(F - 32)$$

Strings

Strings

- A string is a non-primitive pre-defined class in Java. You don't need to import the String class into your programs.
- Strings can be used to hold zero or more characters of data, and can be constructed using the quote ("") delimiters. Example:

```
String message = "Welcome to Java!";
```

Concatenating Strings

- You can concatenate two strings together using the "+" operator. For example

```
String newString = "Hi there, " + "Craig!";
```

- Unlike Python, Java will automatically convert non-String data types into Strings prior to concatenation. For example, the following is a valid statement:

```
String newString = "We are on Chapter #" + 2;
```

Programming Challenge

- Write a program that asks the user to type in 4 different words using the following prompts:

```
enter a noun
enter a verb
enter an adjective
enter an adverb
```

- Use the input to output a "Mad Libs" paragraph using the following text:

```
The [adjective] [noun] was very hungry, so it
decided to [adverb] [verb] to the nearest
restaurant.
```

Programming Challenge

- Write a program that asks the user to enter their name
- Then ask the user to enter in dimensions of their living room in feet
- Calculate the square footage based on their inputted values. Convert this value to meters as well (1 foot = 0.305 meters)

Programming Challenge

- Write a program that asks the user to enter in an integer between 0 and 999
- Add all of the digits together and display the result. For example, the number 854 would yield:

  854 = 8 + 5 + 4 = 17

Programming Challenge

- Write a program that calculates the final score for a student on our class. The student earned the following grades:

```
Homework (25%):
100.0, 95.0, 92.0

Midterms (40%):
89.0,  87.0

Final Exam (35%):
91.5
```