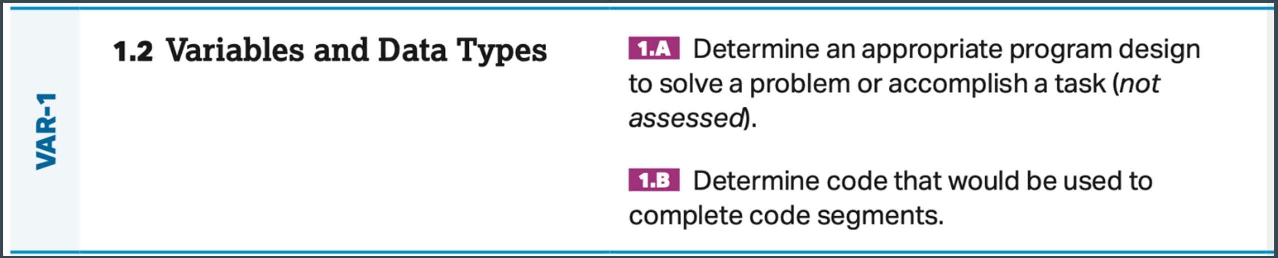
AP Computer Science A

UNIT 1 TOPIC 2
Variables & Data Types

College Board Alignment
Unit 1 Topic 2

UNIT 1

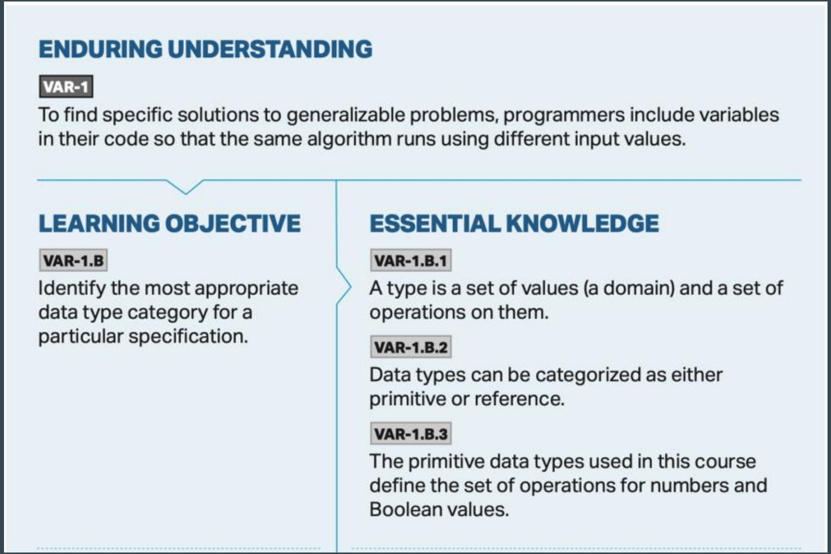
Primitive Types


VAR-1

1.2 Variables and Data Types

1.A Determine an appropriate program design to solve a problem or accomplish a task (*not assessed*).

1.B Determine code that would be used to complete code segments.


ENDURING UNDERSTANDING

VAR-1
To find specific solutions to generalizable problems, programmers include variables in their code so that the same algorithm runs using different input values.

LEARNING OBJECTIVE

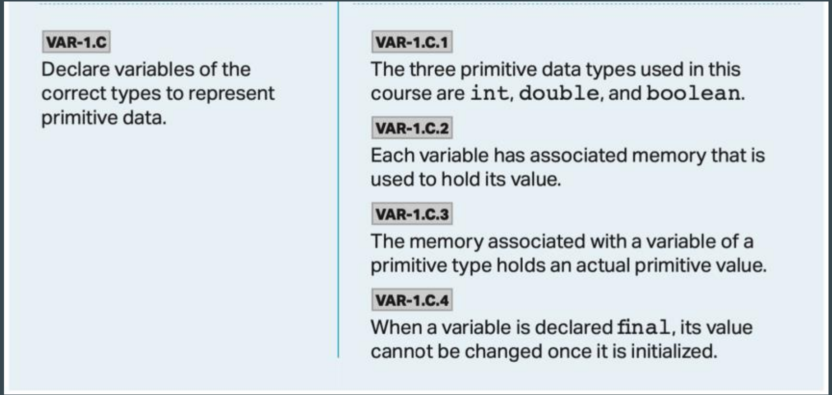
VAR-1.B
Identify the most appropriate data type category for a particular specification.

ESSENTIAL KNOWLEDGE

VAR-1.B.1
A type is a set of values (a domain) and a set of operations on them.

VAR-1.B.2
Data types can be categorized as either primitive or reference.

VAR-1.B.3
The primitive data types used in this course define the set of operations for numbers and Boolean values.


VAR-1.C
Declare variables of the correct types to represent primitive data.

VAR-1.C.1
The three primitive data types used in this course are `int`, `double`, and `boolean`.

VAR-1.C.2
Each variable has associated memory that is used to hold its value.

VAR-1.C.3
The memory associated with a variable of a primitive type holds an actual primitive value.

VAR-1.C.4
When a variable is declared `final`, its value cannot be changed once it is initialized.

Goals for Today

- Understand the different types of data used in Java, including primitive and reference types
- Print literals of other data types to the console
- Understand how to declare and initialize variables in Java
- Identify which variable type is most appropriate for a given situation
- Understand naming conventions for variables
- Use the `final` keyword to create constants

Data Types

Think & Share! From your experience in AP CSP, what different data types do we use in computer programming?

Data Types

- There are two categories of **data types** in Java: **primitive data** and **objects** (objects are often referred to as **reference data**, which we will talk about in Unit 2)

Data Types

- There are two categories of **data types** in Java: **primitive data** and **objects** (objects are often referred to as **reference data**, which we will talk about in Unit 2)
- The 8 **primitive data types** in Java, **three of which you need to know for the AP Exam**:
  - *Numeric* primitives: `short`, `int`, `long`, `float`, `double`
  - *Textual* primitives: `byte`, `char`
  - *Boolean* primitive: `boolean`

Data Types

- There are two categories of **data types** in Java: **primitive data** and **objects** (objects are often referred to as **reference data**, which we will talk about in Unit 2)
- The 8 **primitive data types** in Java, **three of which you need to know for the AP Exam**:
  - *Numeric* primitives: `short`, `int`, `long`, `float`, `double`
  - *Textual* primitives: `byte`, `char`
  - *Boolean* primitive: `boolean`

The three you need to know

Primitive Data Type	Explanation	Examples of literals of this type
`int`	This is the type for **integer** values (i.e. positive/negative whole numbers)	`-71` `-5` `0` `14` `283` `16930`
`double`	This is the type for **floating point** (decimal) values (i.e. real numbers)	`-6.31` `0.5` `3.0` `0.025` `12.378`
`boolean`	This is the type for **true/false** values (there are only two possible values!)	`true` `false`

Data Types

- There are two categories of **data types** in Java: **primitive data** and **objects** (objects are often referred to as **reference data**, which we will talk about in Unit 2)

- The 8 **primitive data types** in Java, **three of which you need to know for the AP Exam**:
  - *Numeric* primitives: `short, int, long,` `float, double`
  - *Textual* primitives: `byte, char`
  - *Boolean* primitive: `boolean`

These are all integer types, but differ in the amount of memory they take up, and the max/min value you can store

These are both **floating point** (decimal) types, but differ in amount of memory they take up and *precision* of decimal stored

The three you need to know

Primitive Data Type	Explanation	Examples of literals of this type
`int`	This is the type for **integer** values (i.e. positive/negative whole numbers)	`-71` `-5` `0` `14` `283` `16930`
`double`	This is the type for **floating point** (decimal) values (i.e. real numbers)	`-6.31` `0.5` `3.0` `0.025` `12.378`
`boolean`	This is the type for **true/false** values (there are only two possible values!)	`true` `false`

Wait, where’s the “string” type?!

What about “Hello world!” and other string literals? What data type is a **string literal**?

- In Java, *string literals are* **`objects`** *of type* **`String`** (**`String`** is the class from which **`String`** objects get created); objects are of the *reference* type (they are not primitive type)

Wait, where’s the “string” type?!

What about “Hello world!” and other string literals? What data type is a **string literal**?

- In Java, *string literals are* **`objects`** *of type* **`String`** (**`String`** is the class from which **`String`** objects get created); objects are of the *reference* type (they are not primitive type)
- As we will see, creating a new *class* effectively creates a brand new *type*. Any time you create a new class, you are actually telling Java, “Hey, here is a new type to use in my program!”

Wait, where's the "string" type?!

What about "Hello world!" and other string literals? What data type is a **string literal**?

- In Java, *string literals are* **`objects`** *of type* **`String`** (**`String`** is the class from which **`String`** objects get created); objects are of the *reference* type (they are not primitive type)
- As we will see, creating a new *class* effectively creates a brand new *type*. Any time you create a new class, you are actually telling Java, "Hey, here is a new type to use in my program!"

Reference Data Type (Objects)	**Explanation**	**Examples of literals of this type** (Reminder: quotes are used for string literals)
`String`	This is the class that creates **String** objects, which hold textual strings	**`"Hello world!"`** **`"45"`** **`"~~BOOM!~~"`** **`"m"`** **`"true"`** **`"slskjdlkj23"`**

Quick check!

Quick Check! Which data type is each literal?

`int`, `double`, `boolean`, `String`, or none of these?

`int`	`264`
`String`	`"eighteen"`
`double`	`-9.99999`
`double`	`4.0`
`boolean`	`true`
`String`	`"35"`
`String`	`"12.07"`
`String`	`"true"`
`String`	`"OK, i get it, geez"`
`None!!`	`False`

Choosing Variable Types & Names

Which data type -- **`int, double, boolean, or String`** -- would be most appropriate for each of the following variables?

Variable name:	To store the value of:	Appropriate data type:
`totalCost`	the total cost of groceries, in dollars and cents	
`nickels`	the number of nickels in a piggy bank	
`playerLastName`	the last name of whoever is playing your game	
`failedTest`	whether or not a test grade is below 65	

Choosing Variable Types & Names

Which data type -- **`int, double, boolean, or String`** -- would be most appropriate for each of the following variables?

Variable name:	To store the value of:	Appropriate data type:
`totalCost`	the total cost of groceries, in dollars and cents	**`double`**
`nickels`	the number of nickels in a piggy bank	**`int`**
`playerLastName`	the last name of whoever is playing your game	**`String`**
`failedTest`	whether or not a test grade is below 65	**`boolean`**

There is a convention (best practice) for naming variables in Java -- using these four variable names, can you determine what that convention is?

Choosing Variable Types & Names

Which data type -- `int, double, boolean, or String` -- would be most appropriate for each of the following variables?

Variable name:	To store the value of:	Appropriate data type:
`totalCost`	the total cost of groceries, in dollars and cents	**`double`**
`nickels`	the number of nickels in a piggy bank	**`int`**
`playerLastName`	the last name of whoever is playing your game	**`String`**
`failedTest`	whether or not a test grade is below 65	**`boolean`**

For naming variables we use the "camelCase" naming convention to name variables: *all variable names **start with lowercase**, then multi-word names use **capital letters to start any subsequent words.***

Printing Literals

Just like with string literals...

We can print out literals of *other* types!

Before we run this code, based on your knowledge about syntax rules and **`println`**, where might you expect compiler errors to occur in the following code? Why?

```
public class Main
{
  public static void main(String[] args)
  {
    System.out.println("a string literal");
    System.out.println(42);
    System.out.println(5.68);
    System.out.println(true);
  }
}
```

prints a **literal** of **type `String`** (aka a "string literal")

prints a **literal** of **type `int`**

prints a **literal** of **type `double`**

prints a **literal** of **type `boolean`**

Think of a literal as simply an *example* of the specific type (see slide 7 for more examples).

DEMO Creating & Using Variables in Java

Step 1. Declare a variable (this "creates" it):

```
type variableName;
```

Step 2. Initialize (to **assign** an initial value to) your variable using the **assignment operator "="**:

```
variableName = value;
```

Step 3. Once you have declared **and** assigned a value to a particular variable, you can use it later in your code, such as in a **`print`** or **`println`** command

```
public class Main
{
  public static void main(String[] args)
  {
    int apples;
    double price;
    String name;
    boolean yummy;

    apples = 10;
    price = 0.75;
    name = "Granny Smith";
    yummy = true;

    System.out.print("Name of apple: ");
    System.out.println(name);
    System.out.print("Number of apples: ");
    System.out.println(apples);
    System.out.print("Price of apple: ");
    System.out.println(price);
    System.out.print("Apple is yummy: ");
    System.out.println(yummy);
  }
}
```

Variable *declaration* ("creating" the variable)

Spaces around operator (not required, but good practice)

Variable *initialization* (assigning an initial value)

Important Things to Know!

You **must** *declare* a variable **before** you try to assign or use it (in other words, in a previous line of code)

If we try to use a variable *before* we declare (create) it, Java will give a compiler error!

If we forget to declare a variable entirely but still try to use it, Java will yell at us! In this case, we haven't declared (created) peaches so it is "illegal" to try and assign it.

When you put a *variable* in a **`print`** or **`println`** (or in *any* method), we *don't* use quotes!

This will print out

```
Number of apples: apples
```

Instead of

```
Number of apples: 10
```

```
public class Main
{
  public static void main(String[] args)
  {
    apples = 10;
    price = 0.75;

    int apples;
    double price;

    peaches = 5;
    7 = apples;

    System.out.print("Number of apples: ");
    System.out.println("apples");
    System.out.print("Price of apple: ");
    System.out.println(price);
  }
}
```

The *value being assigned* **must** go on the *right* side of the = ! This will cause a compiler error!

Important Things to Know!

Java is a **statically typed** language which means that variables are bound to the data type that they are declared and **type checking** occurs at *compile time*.

Example: You can't assign a **`double`** value to a variable declared as **`int`**

Interestingly, you *can* assign an **`int`** value to a variable declared as **`double`** -- this is because Java automatically converts an int to double behind the scenes (which is possible because a double has longer range, i.e. more memory available for it) -- but it doesn't work in reverse, because int is 32-bit and double is 64-bit). In this example, it would be much better to put 2.0 instead of 2, for the sake of clarity!

```
public class Main
{
  public static void main(String[] args)
  {
    int apples;
    double price;

    apples = 10.5;
    price = 2;

    System.out.print("Number of apples: ");
    System.out.println(apples);
    System.out.print("Number of bananas: ");
    System.out.println(bananas);
  }
}
```

Important Things to Know!

The blank lines *in between lines of code* don't matter; inserting blank lines is a stylistic choice and done to improve readability -- where and when to do this comes with practice!

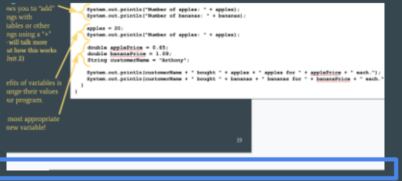
```
public class Main
{
  public static void main(String[] args)
  {
    int apples;
    int bananas;

    apples = 10;
    bananas = 15;

    System.out.print("Number of apples: ");
    System.out.println(apples);
    System.out.print("Number of bananas: ");
    System.out.println(bananas);
  }
}
```

```
public class Main
{
  public static void main(String[] args)
  {
    int apples;
    int bananas;
    apples = 10;
    bananas = 15;
    System.out.print("Number of apples: ");
    System.out.println(apples);
    System.out.print("Number of bananas: ");
    System.out.println(bananas);
  }
}
```

Blank lines ***don't matter*** -- these two pieces of code do **EXACTLY** the same thing!

More on Variables:

Click and drag the slider at the bottom to see the full code



```
public class Main
{
  public static void main(String[] args)
  {
    int apples;
    int bananas;

    apples = 10;
    bananas = 15;

    System.out.print("Number of apples: ");
    System.out.println(apples);
    System.out.print("Number of bananas: ")
    System.out.println(bananas);
  }
}
```

You can **declare *and* initialize** a variable in at the *same time*

String concatenation allows you to combine strings with other strings or variables using "**+**" (we will talk about this more in Unit 2)

One of the benefits of variables is that you can *change* their values as needed in your program.

It is a convention to use "**camelCase**" to name multi-word variables: start with lowercase, then any additional words get a capital first letter

```
public class Main
{
  public static void main(String[] args)
  {
    int apples = 10;
    int bananas = 15;

    System.out.println("Number of apples: " + apples);
    System.out.println("Number of bananas: " + bananas);

    apples = 20;
    System.out.println("Number of apples: " + apples);

    double applePrice = 0.65;
    double bananaPrice = 1.09;
    String customerName = "Anthony";

    System.out.println(customerName + " bought " + apples
    System.out.println(customerName + " bought " + bananas

    boolean raining = true;
    System.out.println("It was raining outside: " + rainin

    String class = "AP CSA";
    System.out.println(class);

    System.out.println(appleprice);
  }
}
```

Spaces around operat
required, but good pra

You *can't* use Java
keywords (like clas
as variable names!

Case sensitivity! applePrice and applepric

More on Variables:

We use the final keyword to indicate a constant variable: a variable that will not (and cannot!) be changed anywhere else in the program

It's convention to have **constants** defined at the *top* of your code (not required, but good practice)

Constants are conventionally named using ALL CAPS with an underscore separating multiple words, rather than camelCase (this is done to make clear that they are constants)

```
public class Main
{
  public static void main(String[] args)
  {
    // constants are conventionally ALL CAPS with an
    // underscore to separate words, rather than camelCase
    final double QUARTER_VALUE = 0.25;
    final double PI = 3.14159;

    System.out.println(QUARTER_VALUE);
    System.out.println(PI);

    PI = 3.14;
  }
}
```

You can't update the value of a constant once it has been initialized (hence, "constant"!); doing so will cause a compile time error

Why should we use the final keyword for constants? Couldn't we just *not* change the variable anywhere? Yes, you could, but doing it this way shows INTENT to other programmers that might read your code!

Summary of variables

- There are two categories of data in Java: **primitive data** and **objects** (objects are often referred to as **reference data**)
- The three **primitive data types** in Java that we will use are: `int` (integers), `double` (decimals, i.e. real numbers), and `boolean` (`true` and `false`)
- String literals are `objects` created from the class `String`; they are *not* primitive data.
- To create a **variable** in Java, you **declare** it this way: `type variableName;`
- After declaring a variable, you **initialize** it (assign an initial value to) by using the **assignment operator**, **=**, in this way: `variableName = value;`
- You can declare *and* initialize in one line: `type variableName = value;`
- **String concatenation** allows you to combine strings and variables inside of `print` or `println` statements using "+"