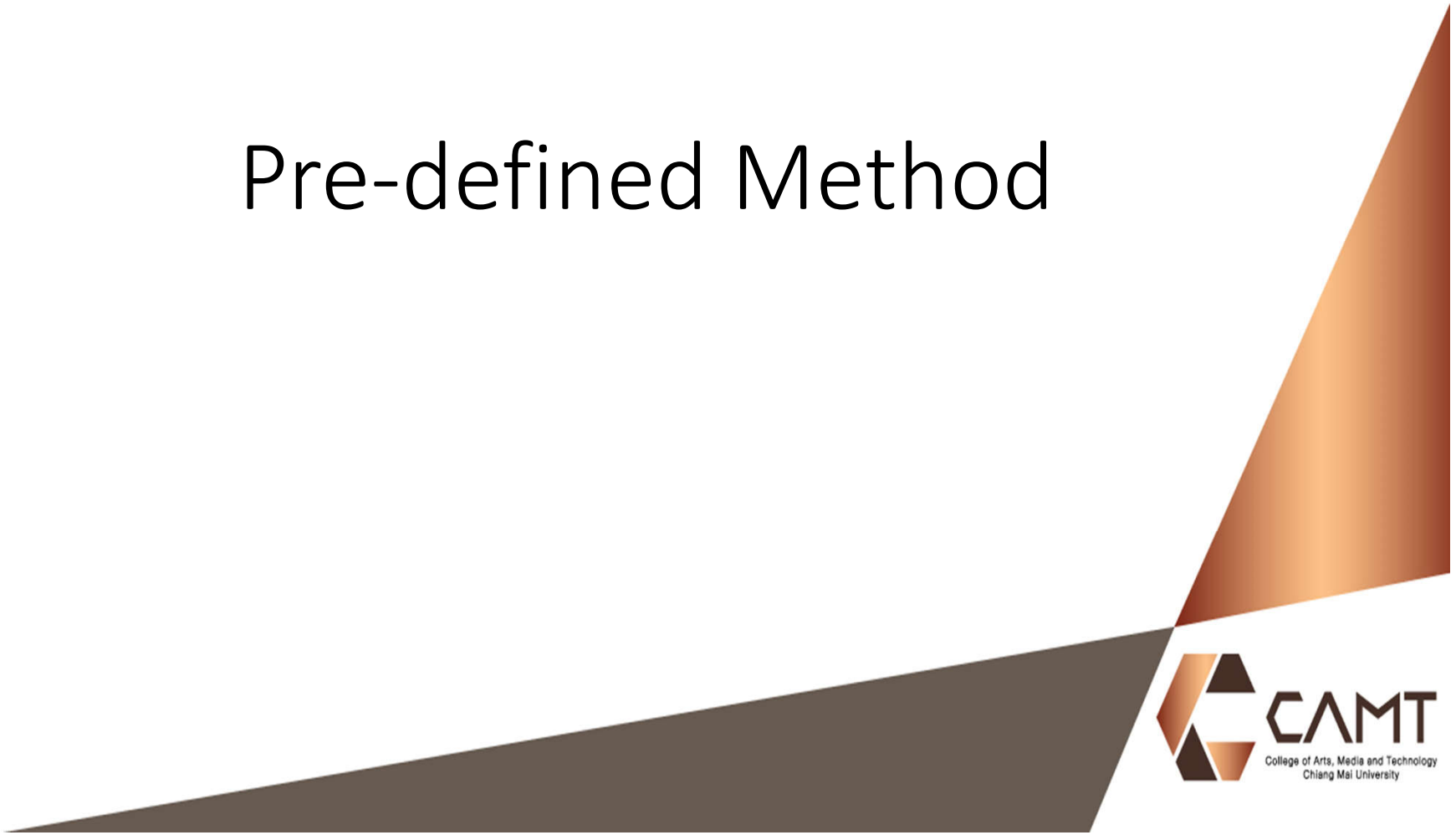
Pre-defined Method



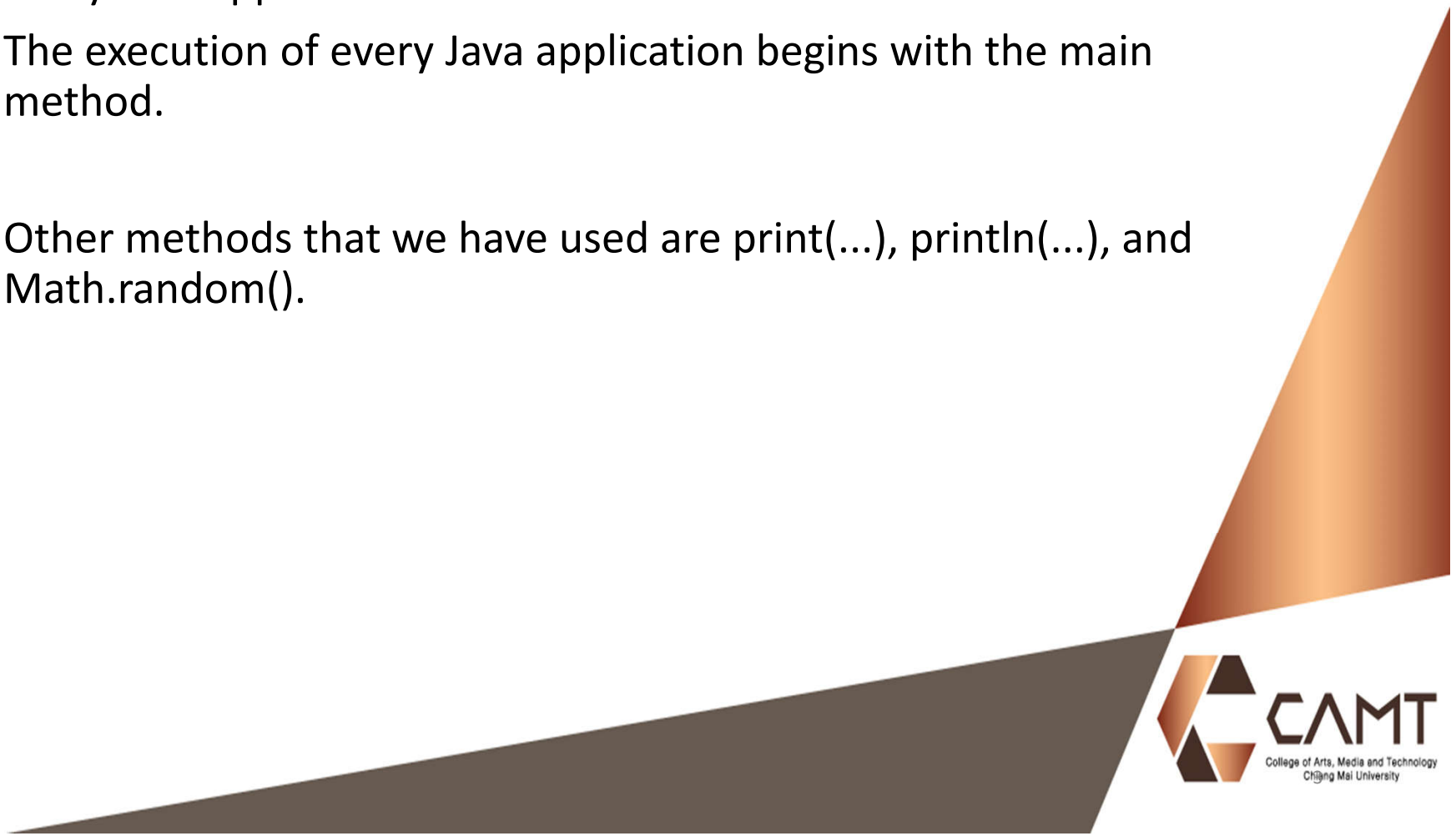
Methods

- A *method* is a named sequence of instructions that are grouped together to perform a task.
- Methods enable the programmer to organize various tasks into neat manageable independent *bundles* of code
- Every Java application that we have written contains one method
  - its name is *main*
  - its instructions appear between the opening and closing braces of main.

Methods

- Every Java application must have a *main* method
- The execution of every Java application begins with the main method.

- Other methods that we have used are print(...), println(...), and Math.random().



Java's Pre-defined Methods

- Imagine a mathematical "*black box*" that works in such a way that whenever you supply a number to the box
  - The box gives or returns the positive square root of that number.

A square root box

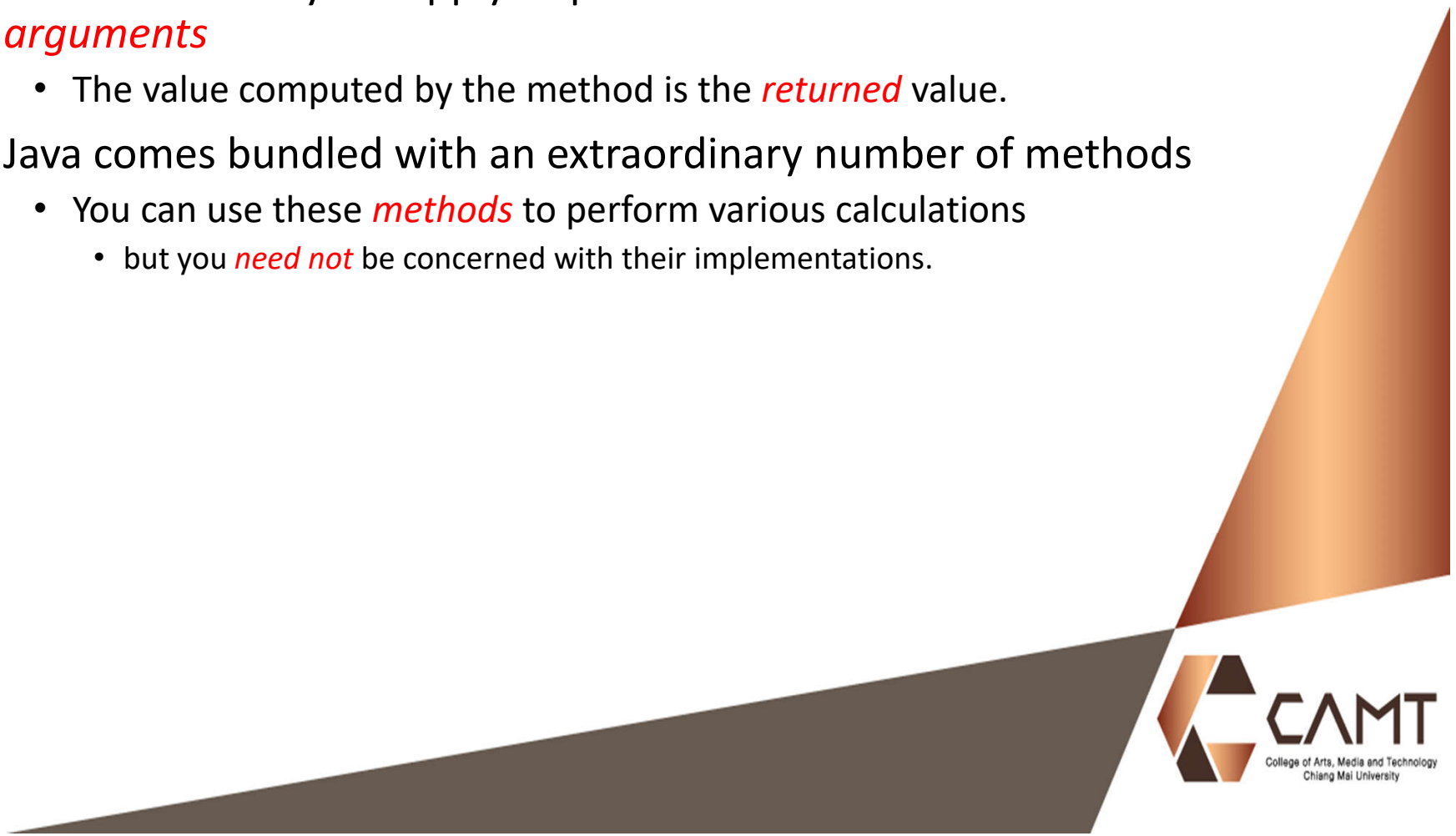
- A similar mechanism that accepts *two* numbers, the *length* and *width* of a rectangle, and returns the area of the rectangle.

An area box

- Such a "*box*" is a metaphor for a method
- A method is very much like a mathematical function
  - a black box that computes an *output* given some *inputs*.

Java's Pre-defined Methods

- The values that you supply or pass to the method are called *arguments*
  - The value computed by the method is the *returned* value.
- Java comes bundled with an extraordinary number of methods
  - You can use these *methods* to perform various calculations
    - but you *need not* be concerned with their implementations.



The Square Root Method

- Example:

In general, the distance to the horizon (in miles) can be estimated as follows:

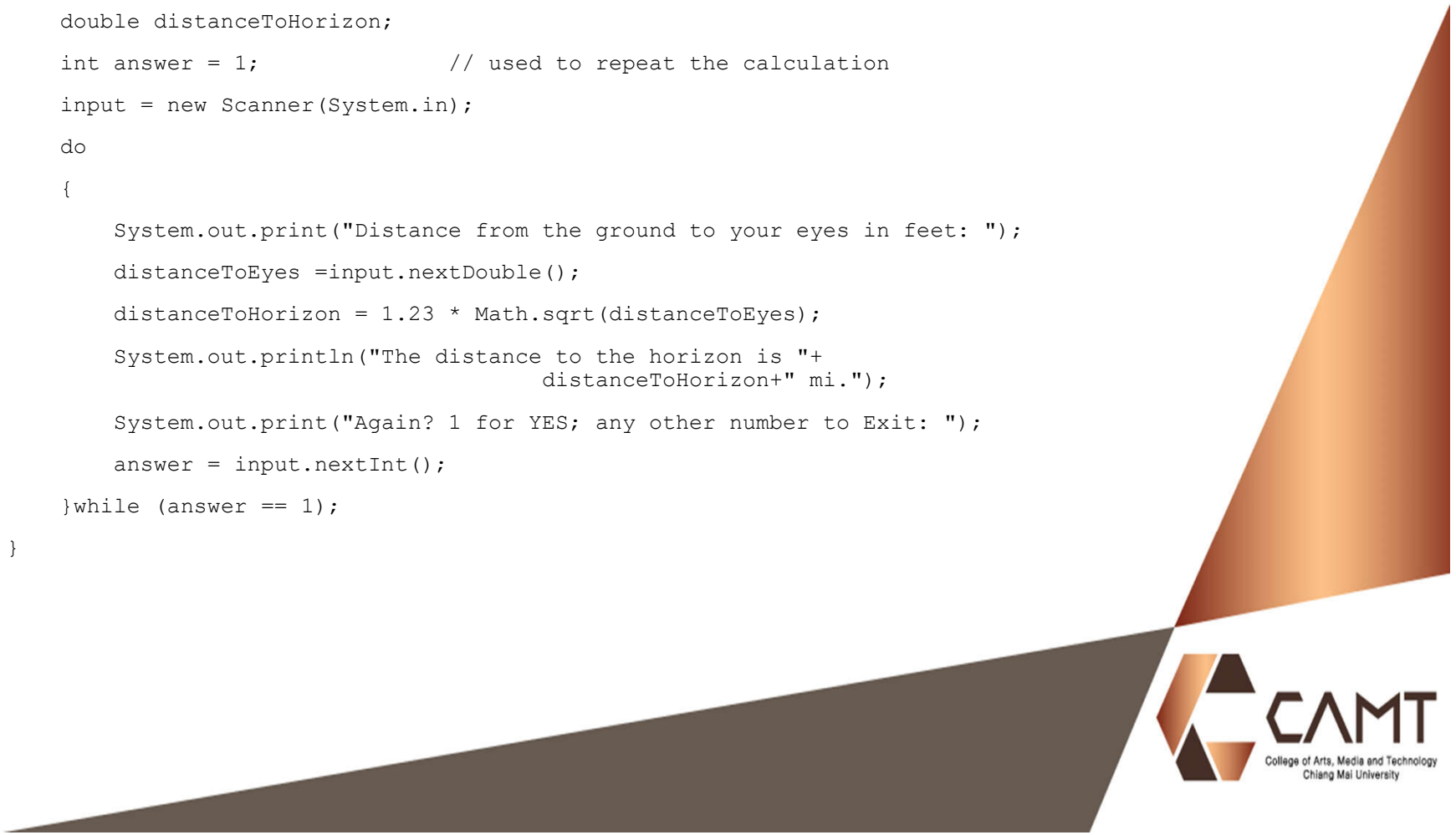
Determine the distance (in feet) from sea level to your eyes.
Compute the square root of that distance.
Multiply the result by 1.23.

The Square Root Methods

- Problem statement:
-

Write a program that prompts a user for the distance measured from the ground to his/her eyes and calculates the distance to the horizon.

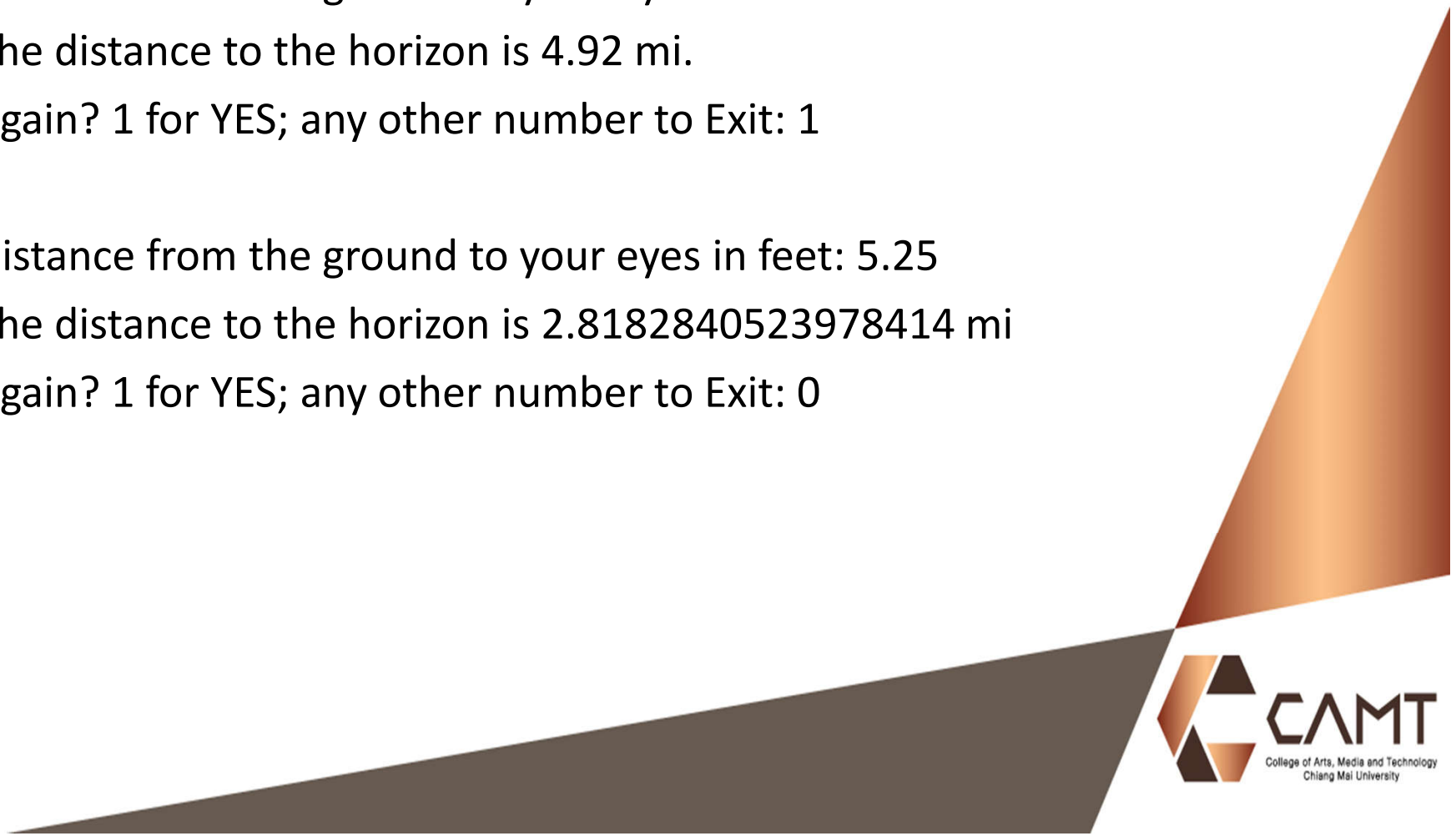
```
import java.util.*;
public class DistanceToHorizon
{
    public static void main(String[] args)
    {
        Scanner input;
        double distanceToEyes;      // measured from the ground
        double distanceToHorizon;
        int answer = 1;              // used to repeat the calculation
        input = new Scanner(System.in);
        do
        {
            System.out.print("Distance from the ground to your eyes in feet: ");
            distanceToEyes =input.nextDouble();
            distanceToHorizon = 1.23 * Math.sqrt(distanceToEyes);
            System.out.println("The distance to the horizon is "+
                                        distanceToHorizon+" mi.");
            System.out.print("Again? 1 for YES; any other number to Exit: ");
            answer = input.nextInt();
        }while (answer == 1);
    }
}
```



Output

Distance from the ground to your eyes in feet: 16.0
The distance to the horizon is 4.92 mi.
Again? 1 for YES; any other number to Exit: 1

Distance from the ground to your eyes in feet: 5.25
The distance to the horizon is 2.8182840523978414 mi
Again? 1 for YES; any other number to Exit: 0



Discussion

- The program utilizes the method :

```
double Math.sqrt(double x)
```

- to calculate the square root of `distanceToEyes`.

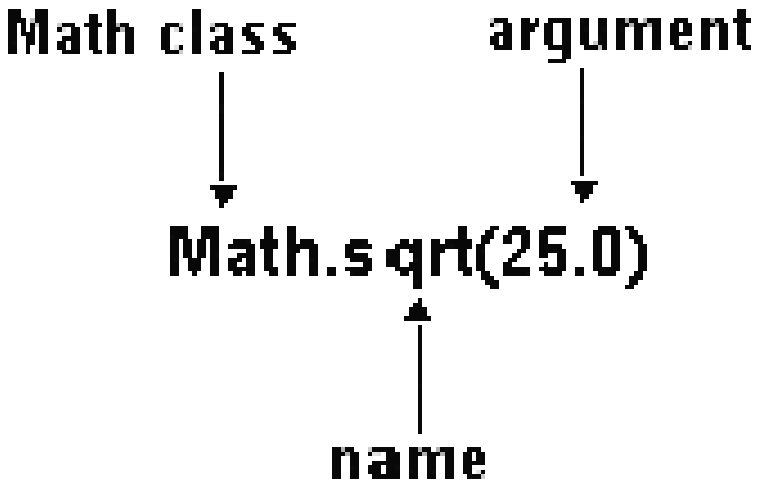
- The method `Math.sqrt(…)` hides the details of its implementation.

Discussion

- How the square root of a number is calculated is *hidden* from the programmer.
- The argument passed to the method is `distanceToEyes`,
  - the returned value is the square root of `distanceToEyes`.
- If `distanceToEyes` has the value 16.0, then `Math.sqrt(distanceToEyes)` returns the value 4.0 and that value is used in the expression

```
distanceToHorizon = 1.23 * Math.sqrt(distanceToEyes);
```

Math.sqrt(double x)



Notice the period that separates the class name Math from the method, sqrt.

The Square Root Method

In the statement

```
double root = Math.sqrt(25.0)
```

- the Math.sqrt(…) method is *called* (or *invoked*) with the argument 25.0 and *returns* the value 5.0 (the square root of 25.0) which is subsequently assigned to the variable root.

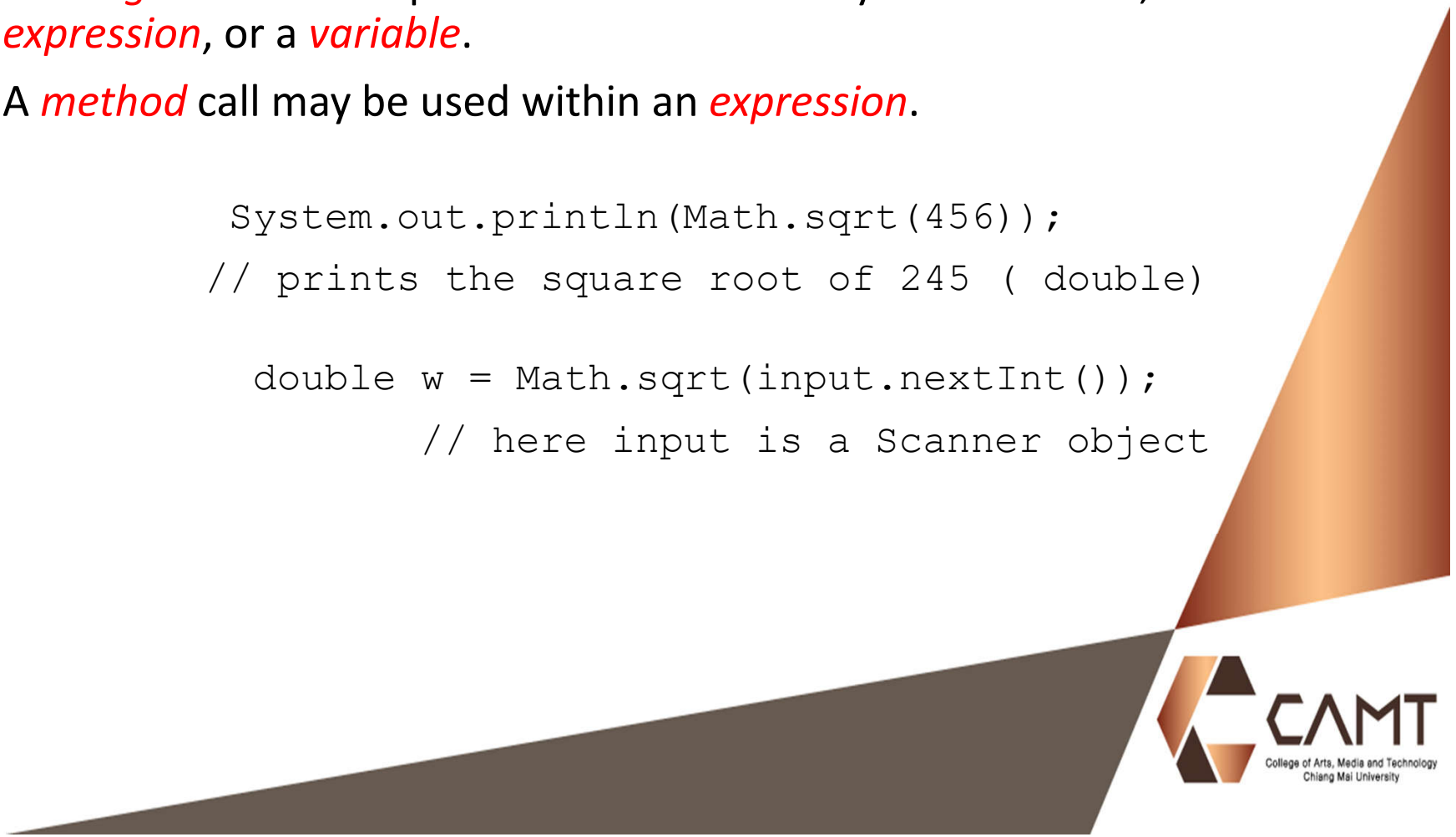
- This action is *similar* to that of the statement:
- ```
  double sum = 5.0 + 8.0;
  ```
  The expression 5.0 + 8.0 evaluates to (or returns)13.0
  - which is assigned to sum.


  ```

- The *argument* that is passed to a method may be a *constant*, an *expression*, or a *variable*.
- A *method* call may be used within an *expression*.

```
System.out.println(Math.sqrt(456));
// prints the square root of 245 ( double)

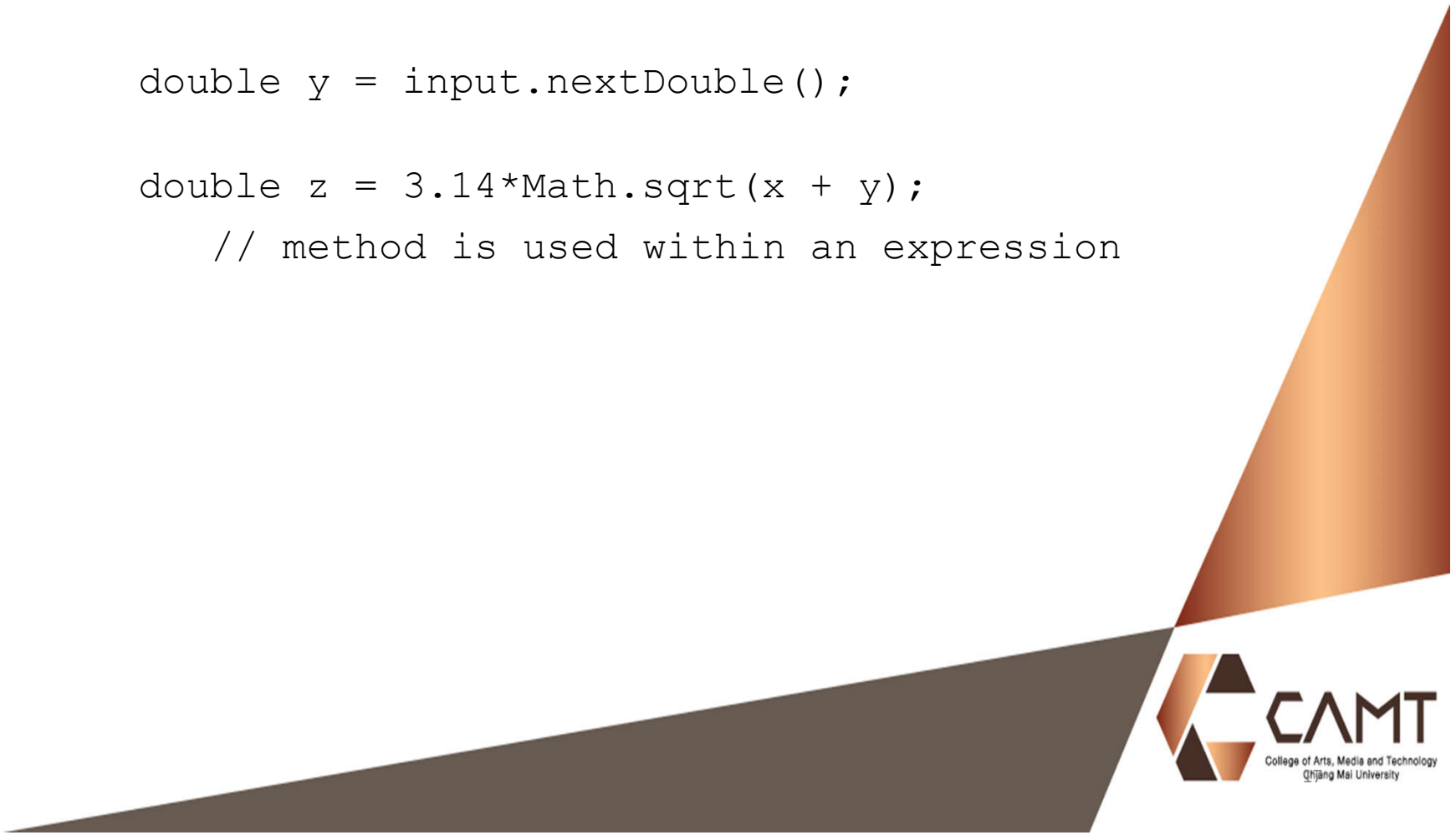
double w = Math.sqrt(input.nextInt());
        // here input is a Scanner object
```



```
double x = input.nextDouble();

double y = input.nextDouble();

double z = 3.14*Math.sqrt(x + y);
   // method is used within an expression
```

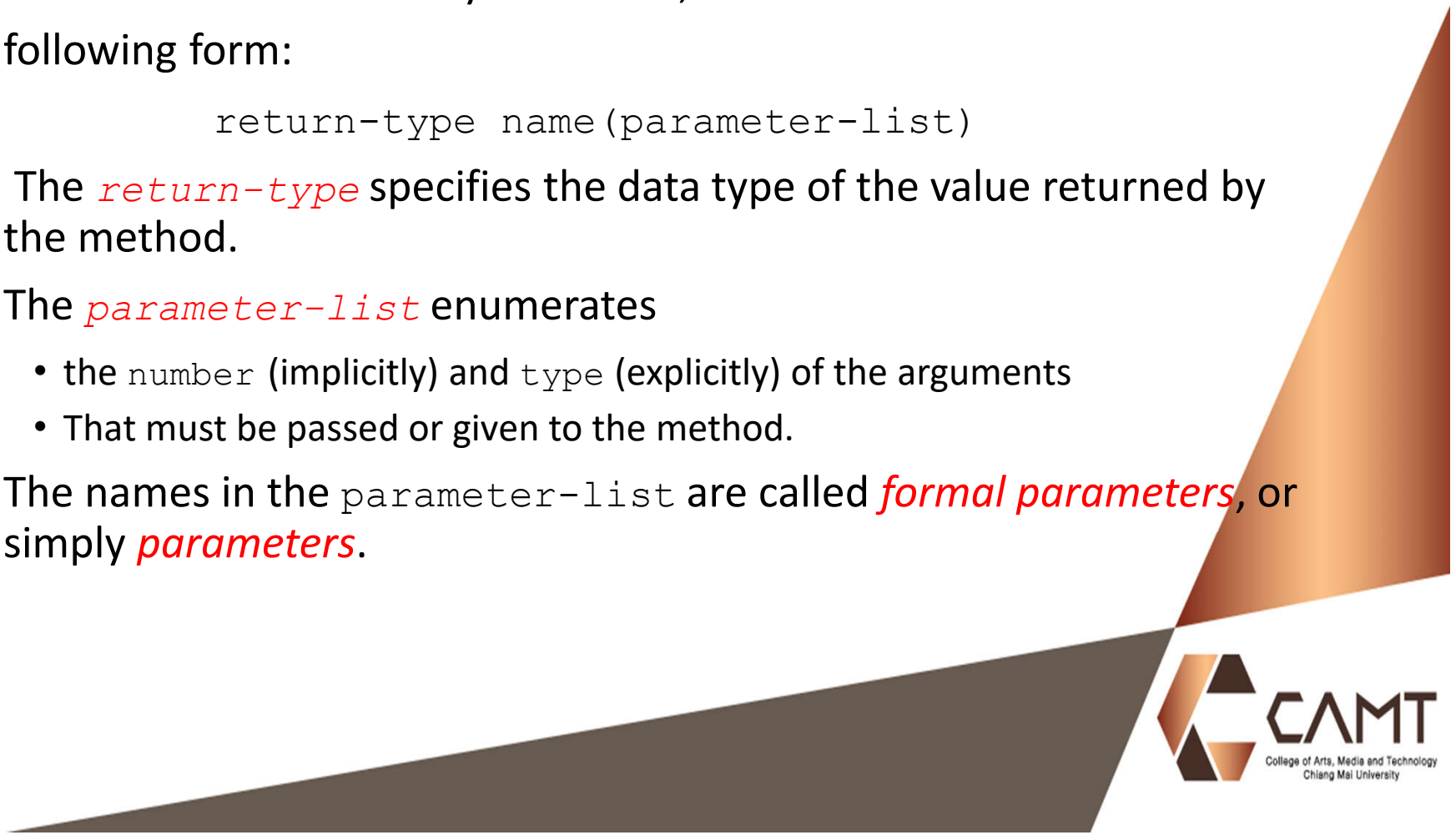


Methods

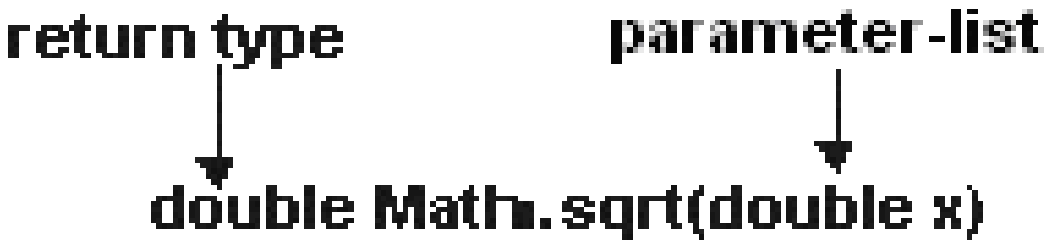
- A method is described by its header, which has the
- following form:

```
return-type name(parameter-list)
```

- The *`return-type`* specifies the data type of the value returned by the method.
- The *`parameter-list`* enumerates
  - the `number` (implicitly) and `type` (explicitly) of the arguments
  - That must be passed or given to the method.
- The names in the `parameter-list` are called *formal parameters*, or simply *parameters*.



Methods



The *header* for Math.sqrt(…)

A Method that Computes Powers

```
double Math.pow(double x, double y)
```

returns x^y.

The parameter list of the header specifies that the method requires two *arguments* of type double.

For example,

`Math.pow(5.0,2.0)` returns $5.0^{2.0}$, i.e., 25.0.

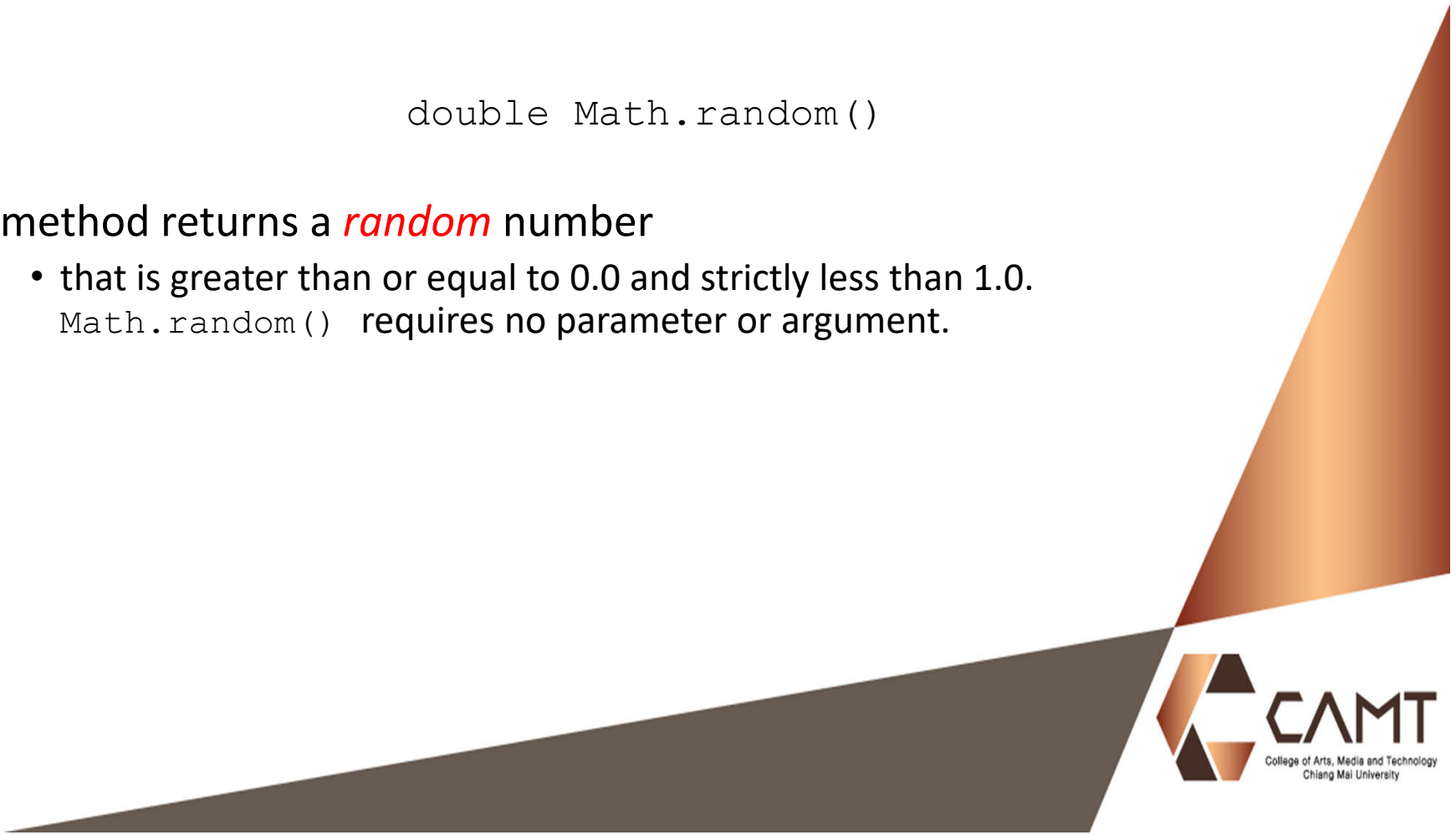
The power method, Math.pow(...)

Random Numbers

- The

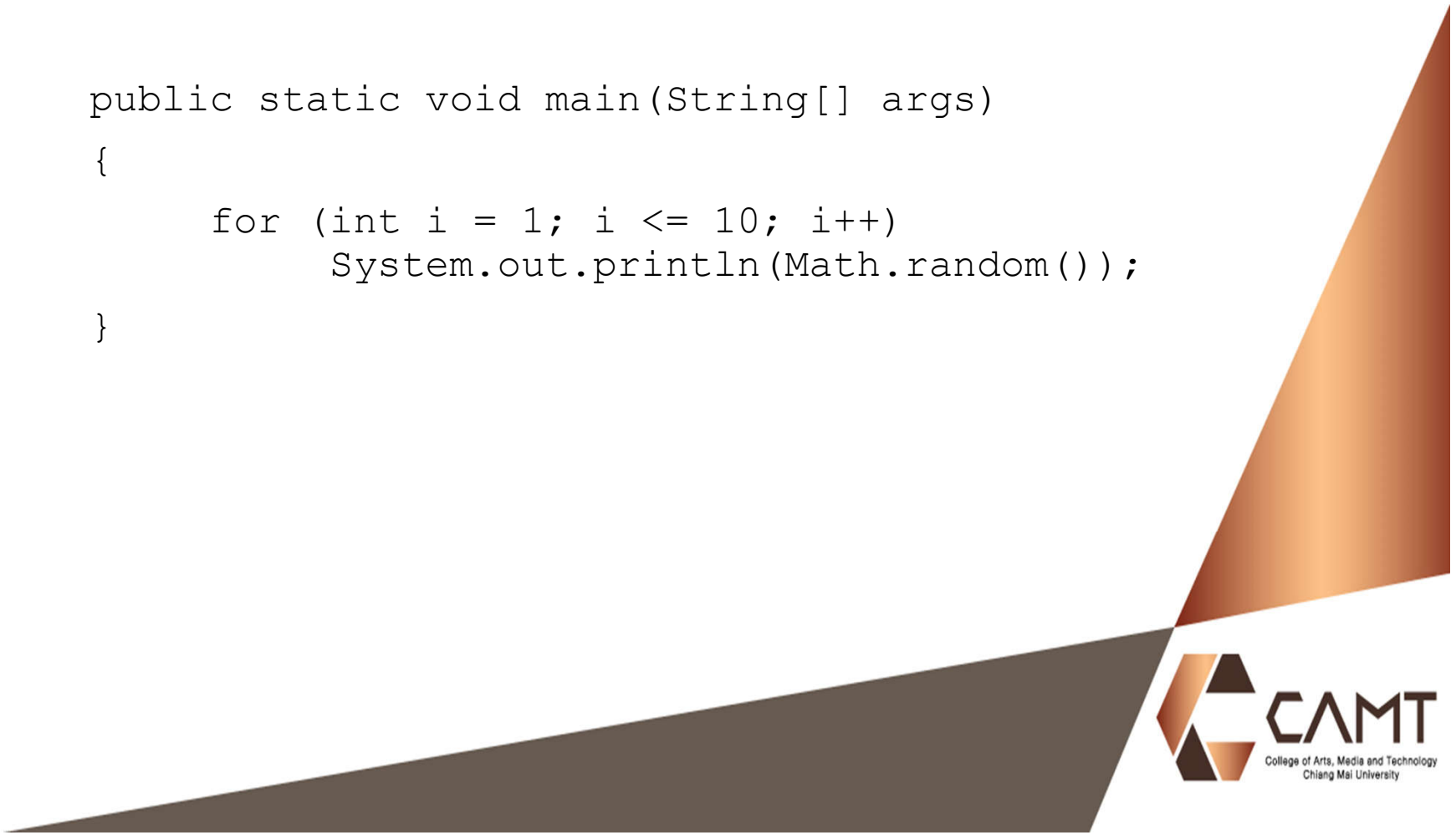
```
double Math.random()
```

- method returns a *random* number
  - that is greater than or equal to 0.0 and strictly less than 1.0. `Math.random()` requires no parameter or argument.



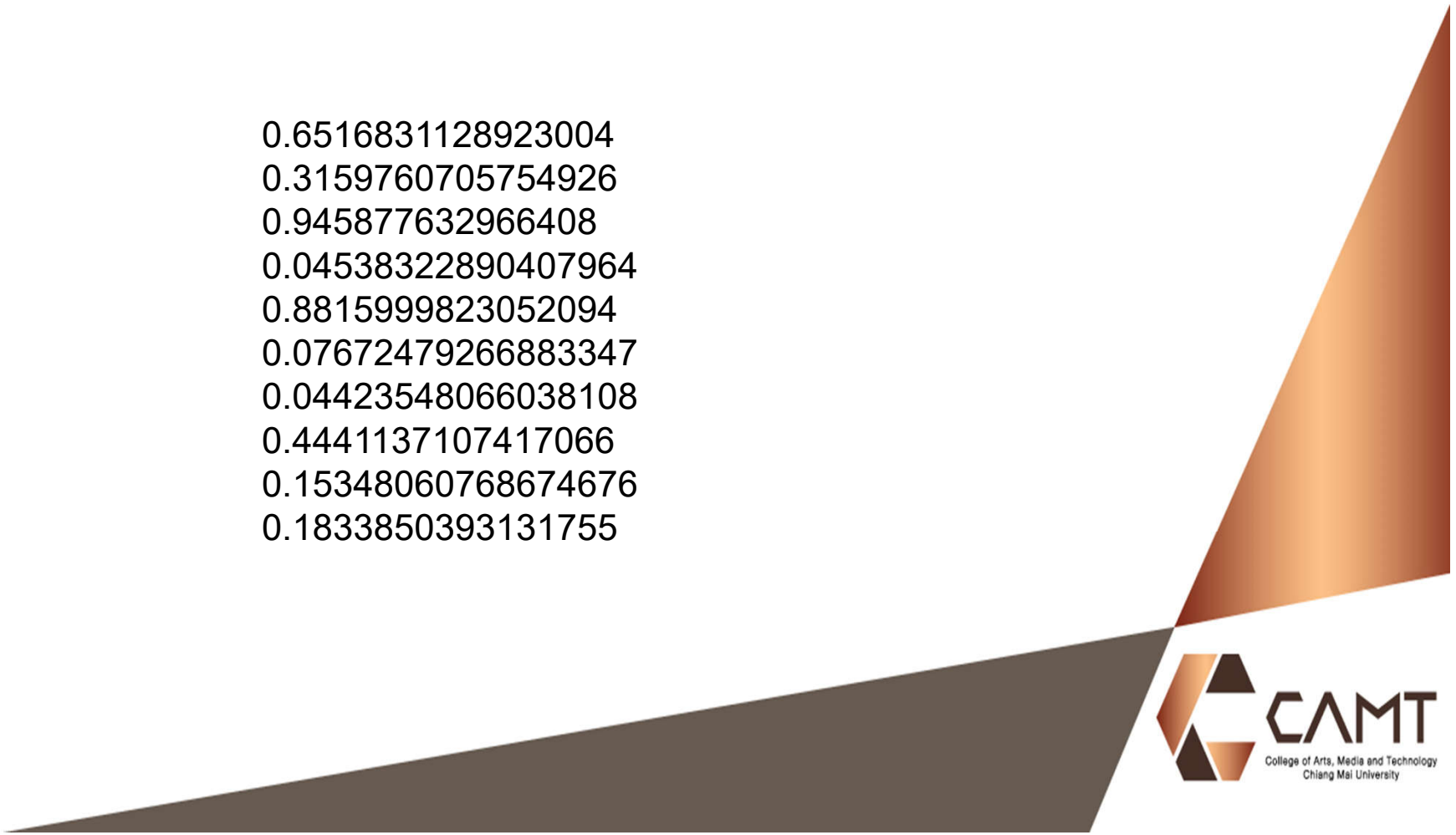
Generate 10 random numbers

```
public class TenRandomNumbers
{
     public static void main(String[] args)
     {
          for (int i = 1; i <= 10; i++)
               System.out.println(Math.random());
     }
}
```



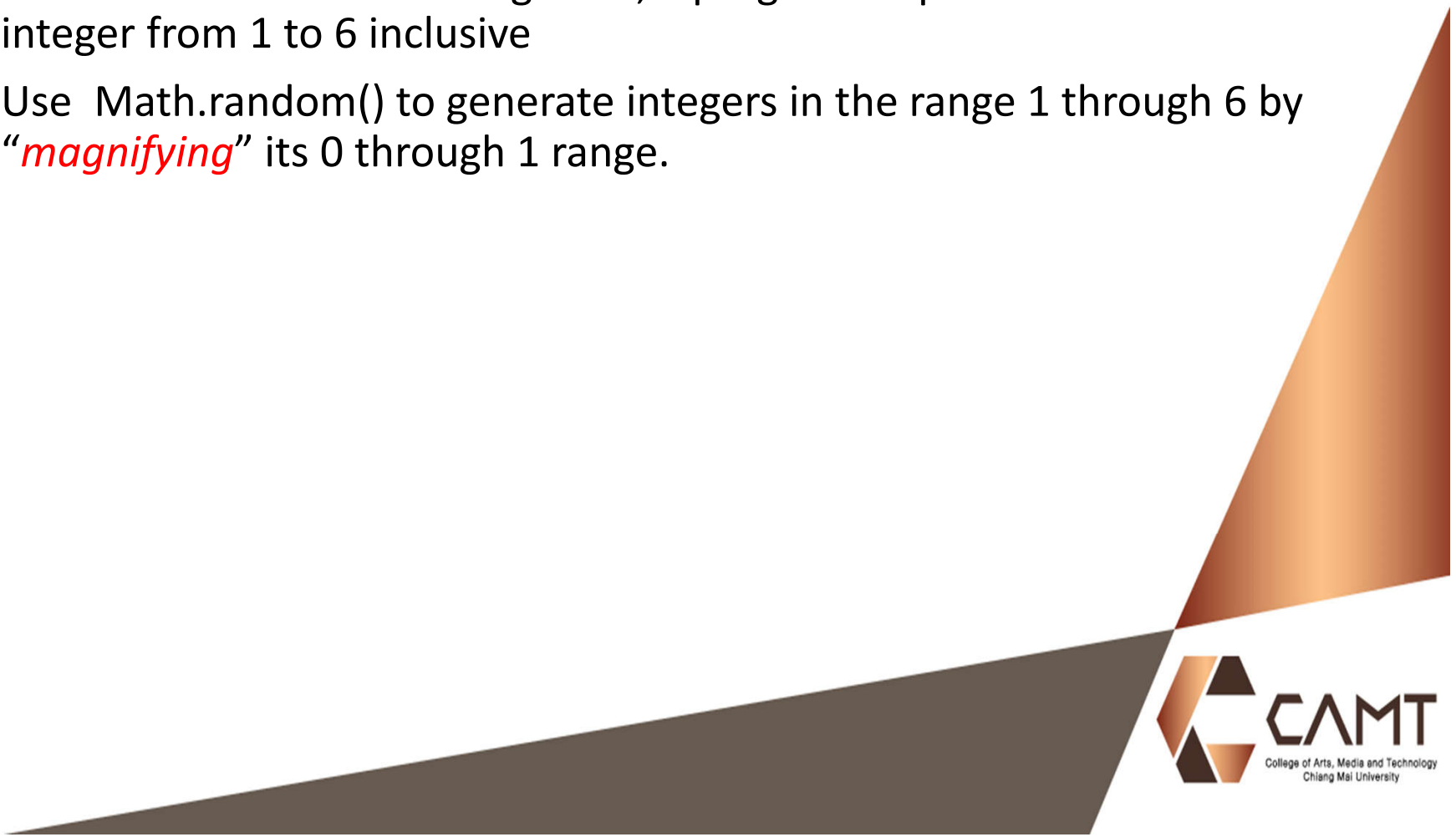
Output

0.6516831128923004
0.3159760705754926
0.945877632966408
0.04538322890407964
0.8815999823052094
0.07672479266883347
0.04423548066038108
0.4441137107417066
0.15348060768674676
0.1833850393131755



Using Math.random() to Generate Integers

- To simulate the roll of a single die, a program requires a random integer from 1 to 6 inclusive
- Use Math.random() to generate integers in the range 1 through 6 by "*magnifying*" its 0 through 1 range.



- Suppose

```
r = Math.random();
```

- Then

  0.0 <= r < 1.0.

  0.0 <= 6*r < 6.0 (multiplying the inequality by 6), and

  1.0 <= 6*r +1 < 7.0. (adding 1 to each value in the inequality)

- Thus

  6*Math.random() + 1 is a number greater than or equal to 1 but strictly less than 7.

Using Math.random() to Generate Integers

- For example, if :

  r = 0.892934399386125 3, then

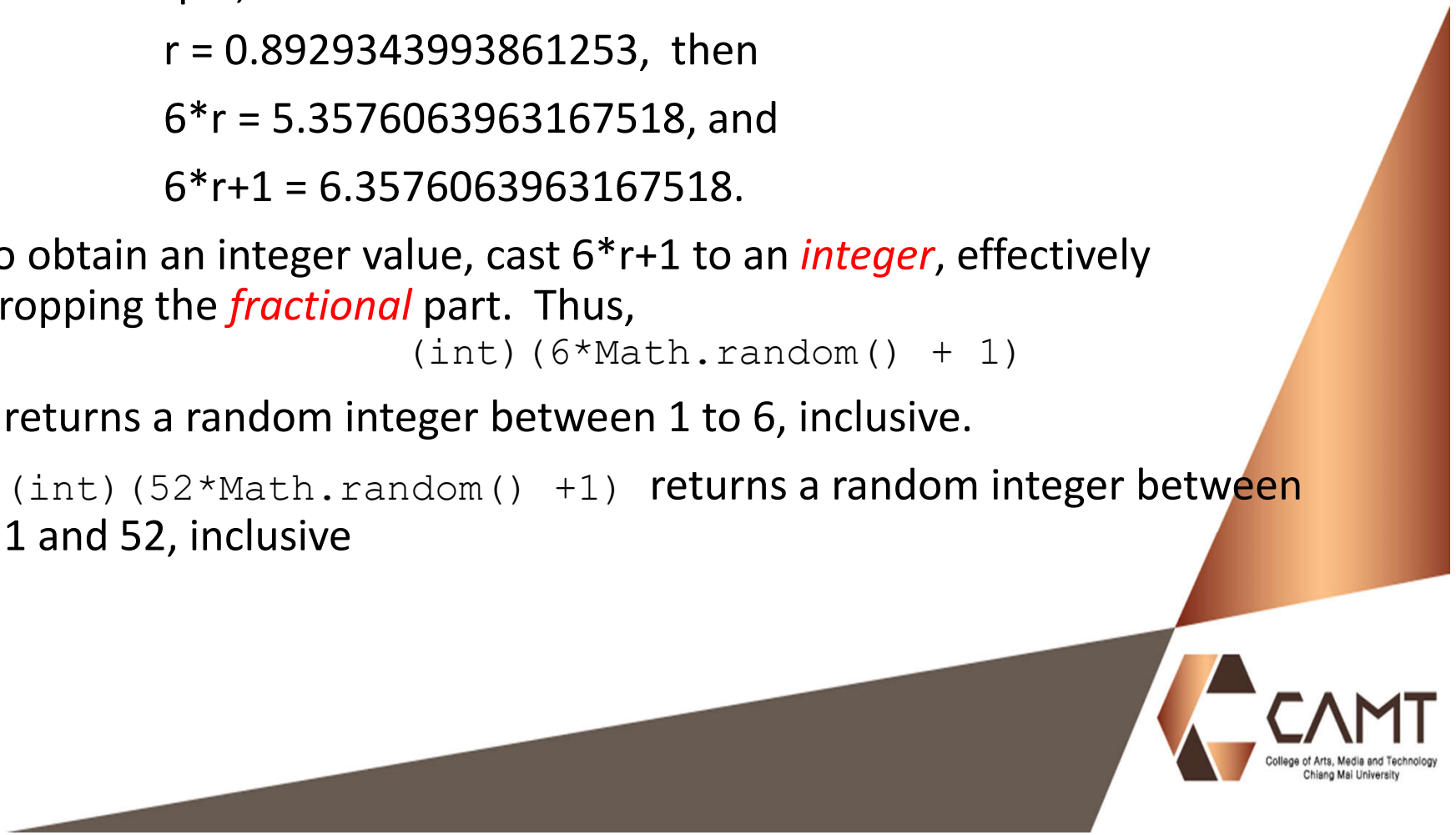
  6*r = 5.357606396316751 8, and

  6*r+1 = 6.357606396316751 8.

To obtain an integer value, cast 6*r+1 to an *integer*, effectively dropping the *fractional* part. Thus,

```
(int)(6*Math.random() + 1)
```

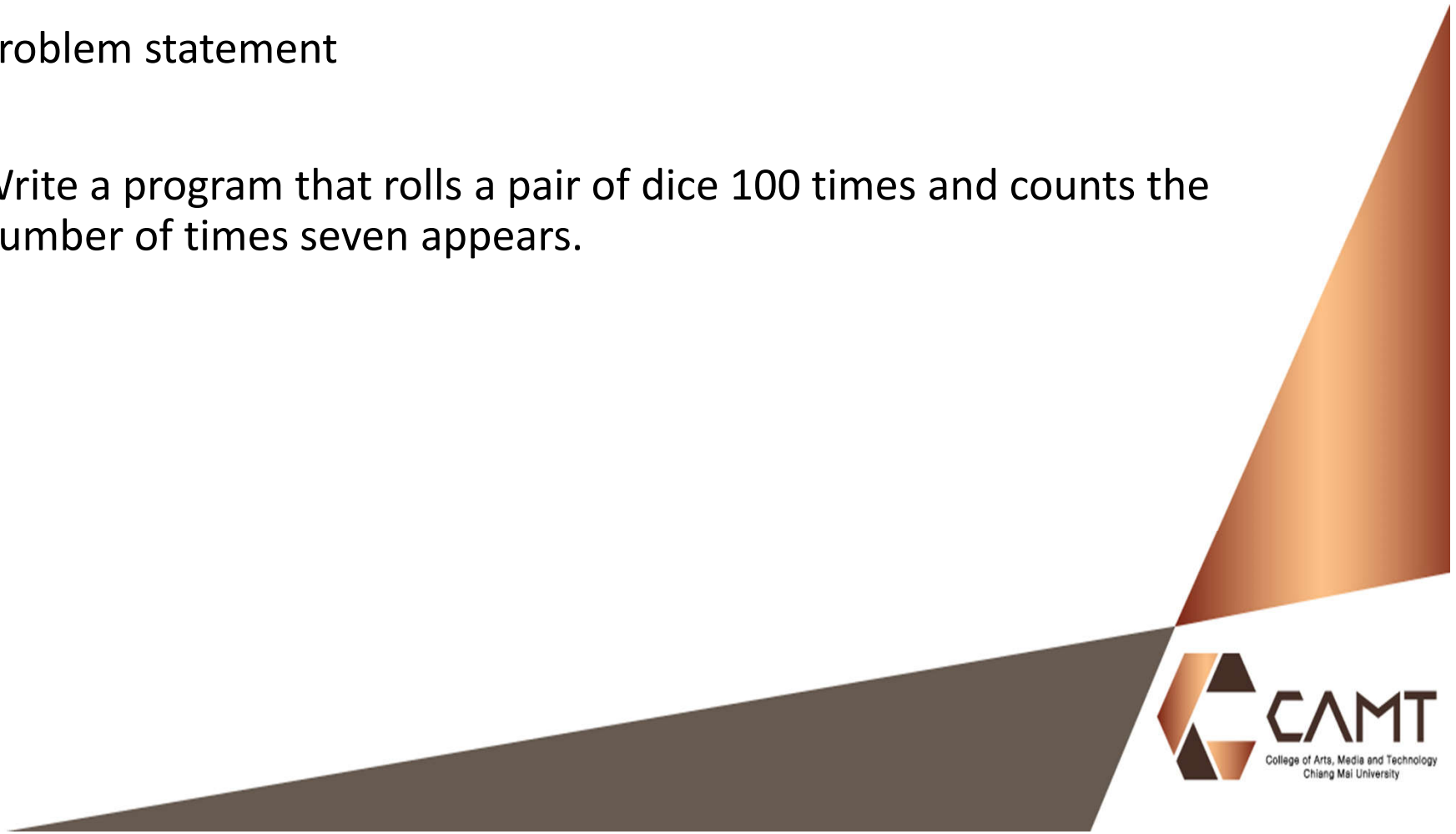
- returns a random integer between 1 to 6, inclusive.
- `(int)(52*Math.random() +1)` returns a random integer between 1 and 52, inclusive



Using Math.random() to Generate Integers

Problem statement

Write a program that rolls a pair of dice 100 times and counts the number of times seven appears.

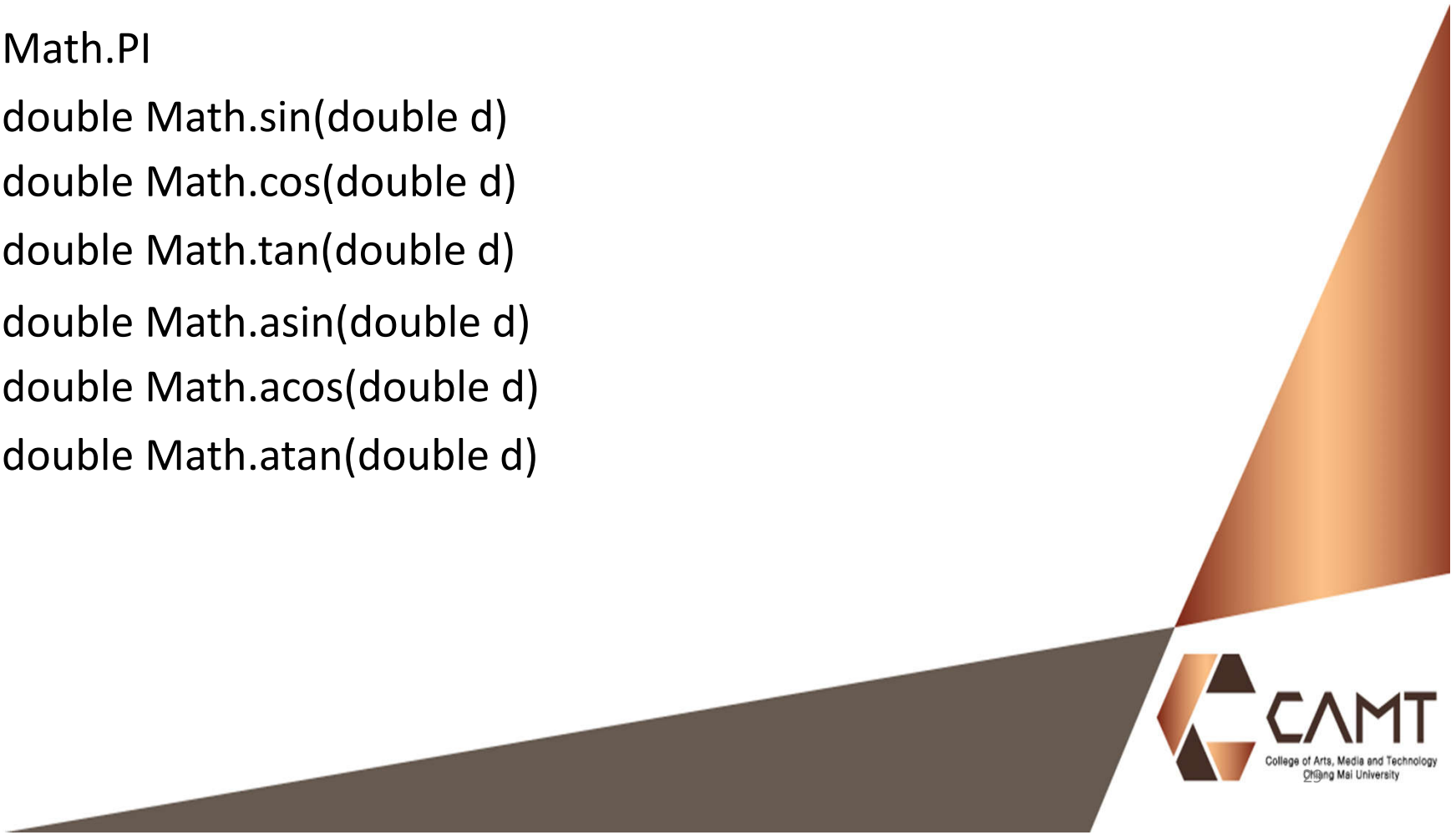


Using Math.random() to Generate Integers

```
import java.util.*;
public class Dice
{
    public static void main(String [] args)
    {
        int die1,die2;
        int sum, seven = 0;
        for (int i = 1; I <= 100; i++)
         {
            die1 = (int)(6*Math.random()+1) ; // random integer 1..6
            die2 = (int)(6*Math.random()+ 1);
            sum = die1 + die2;
            if (sum == 7)
                 seven = seven + 1;
        }
            System.out.println("The number of sevens is " + seven);
    }
}
```

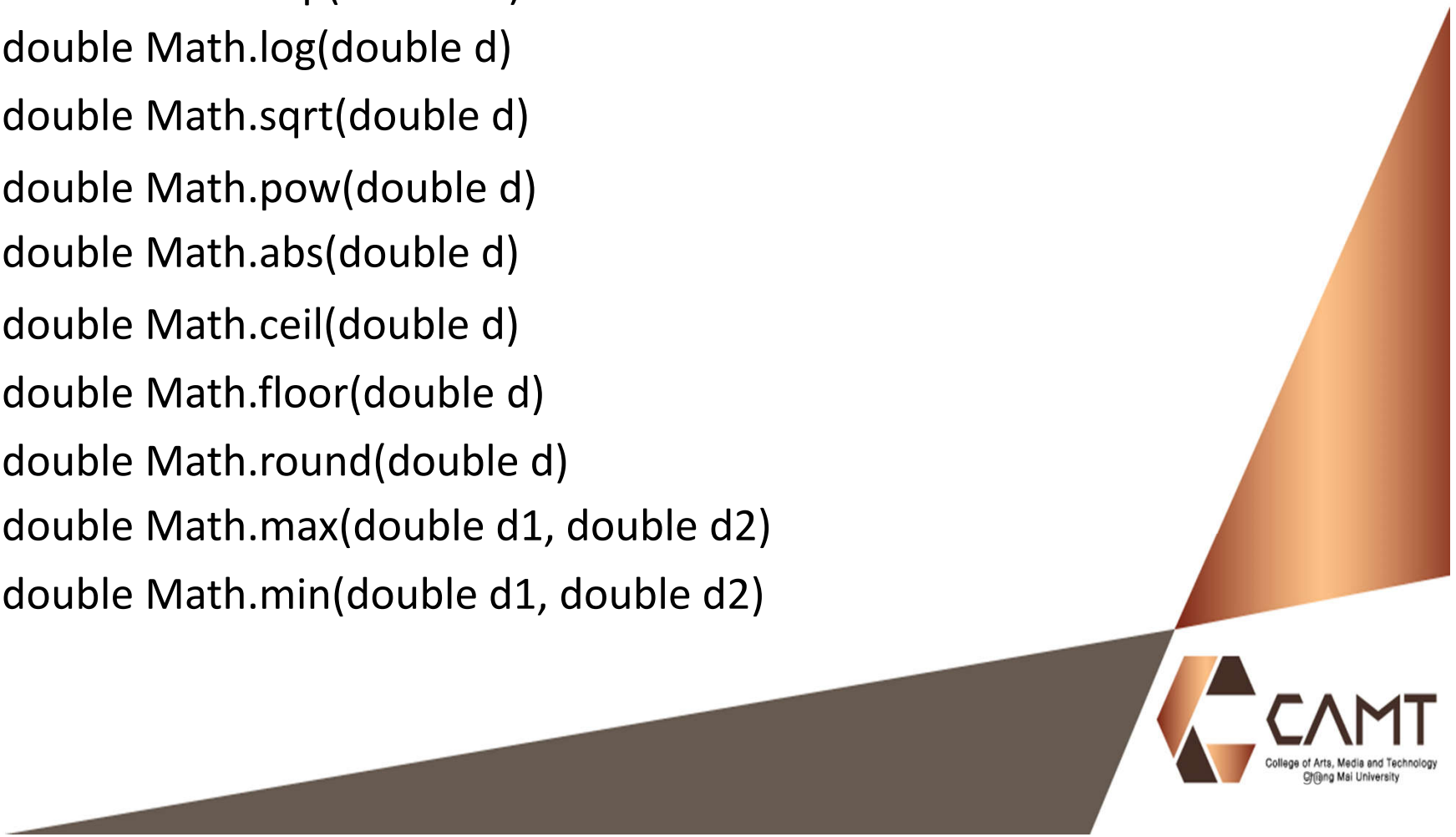
Common function in Mathematic

- Math.E
- Math.PI
- double Math.sin(double d)
- double Math.cos(double d)
- double Math.tan(double d)
- double Math.asin(double d)
- double Math.acos(double d)
- double Math.atan(double d)



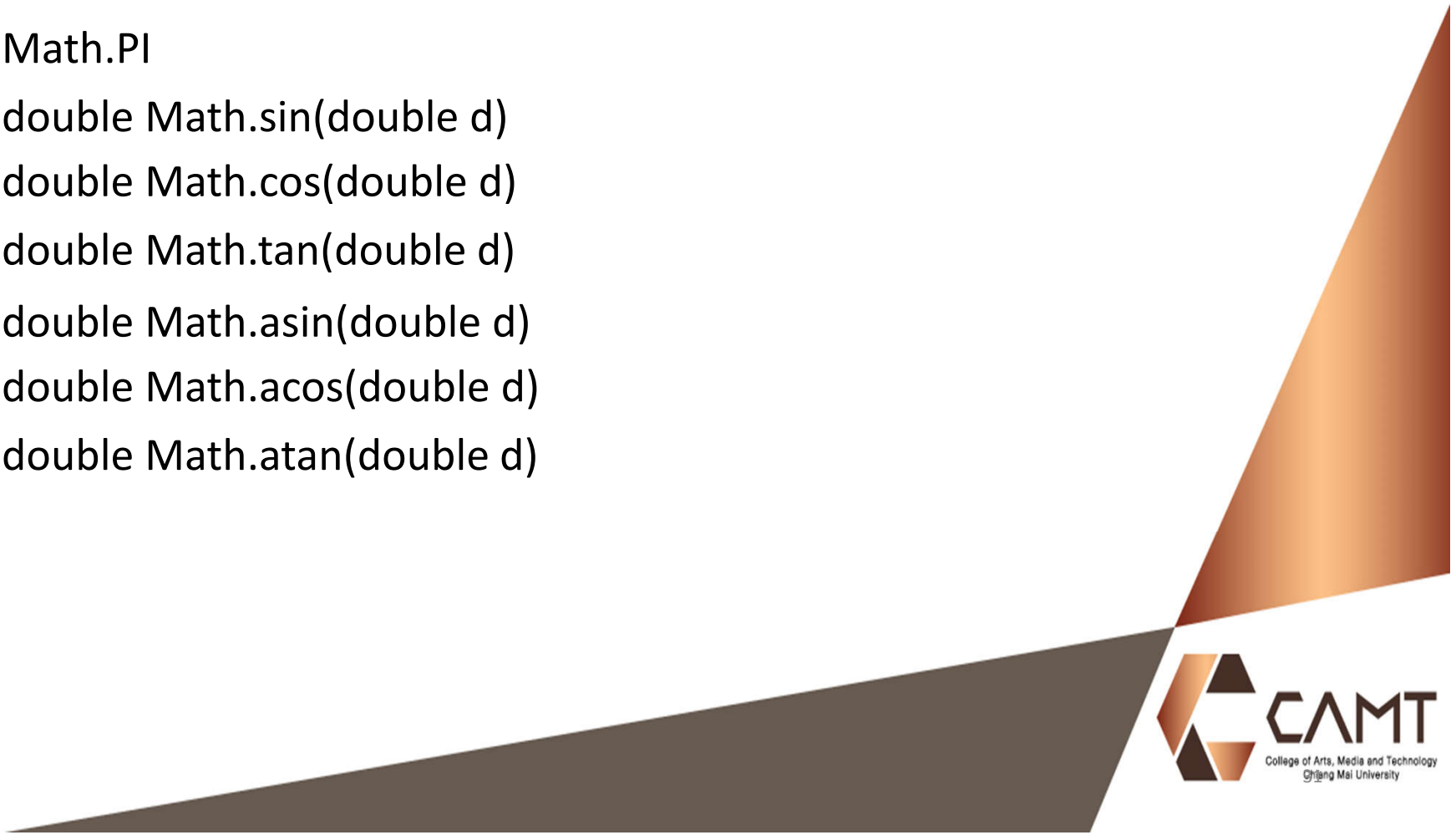
Common function in Mathematic

- double Math.exp(double d)
- double Math.log(double d)
- double Math.sqrt(double d)
- double Math.pow(double d)
- double Math.abs(double d)
- double Math.ceil(double d)
- double Math.floor(double d)
- double Math.round(double d)
- double Math.max(double d1, double d2)
- double Math.min(double d1, double d2)



Common function in Mathematic

- Math.E
- Math.PI
- double Math.sin(double d)
- double Math.cos(double d)
- double Math.tan(double d)
- double Math.asin(double d)
- double Math.acos(double d)
- double Math.atan(double d)



Common function in String

- public boolean equals(Object anObject)
- public boolean equalsIgnore(String anotherString)
- public int compareTo(String anotherString)
- public boolean startsWith(String prefix)
- public boolean endsWith(String suffix)
- public int indexOf(int ch)
- public int indexOf(String str)
- public int lastIndexOf(int ch)
- public int lastIndexOf(String str)
- public char charAt(int index)
- public String substring(int begin)

Common function in String

- public String substring(int begin)
- public String substring(int begin, int end)
- public String replace(char oldChar, char newChar)
- public String toLowerCase()
- public String toUpperCase()