**Data Variables – Java Identifiers**

The general rules for constructing names for variables (unique identifiers) are:

* Case Sensitive
* Begin with letter
* Contains letters, digits, underscore, and dollar signs.
* No Special Characters

**Data types are divided into two groups:**

* Primitive data types - includes byte, short, int, long, float, double, Boolean, and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp)

1. The byte data type can store whole numbers from -128 to 127. **(SIZE: 1 Byte).**
2. The short data type can store whole numbers from -32768 to 32767. **(SIZE: 2 Byte).**
3. The int data type can store whole numbers from -2147483648 to 2147483647. **(SIZE: 4 Byte).**
4. The long data type can store whole # from -9223372036854775808 to 9223372036854775807. **(SIZE: 8 Byte).**
5. The float data type can store fractional numbers from 3.4e−038 to 3.4e+038. **(SIZE: 4 Byte).**
6. The double data type can store fractional numbers from 1.7e−308 to 1.7e+308. **(SIZE: 8 Byte).**
7. A boolean data type is declared with the boolean keyword & can take the values true or false: **(SIZE: 1 bit).**
8. The char data type is used to store a **single** character. **(SIZE: 2 Byte).**

* The String data type is used to store a sequence of characters (text).

A **String in Java** is actually a non-primitive data type, because it refers to an object. The String object has methods that is used to perform certain operations on strings

**Primitive Data Types VS Non Primitive Data Types**

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
* The size of a primitive type depends on the data type, while non-primitive types have all the same size.

**Code Snippets:**

int z = 5;

int y = 5;

double power = Math.pow(z,y);

System.out.println(power);

Output: 3125.0

Math.pow datatype has to be Double. Conversion error if it is an integer.

// Create a Scanner Object

Scanner sc = new Scanner(System.in);

System.out.print("Please Enter the Input: ");

// It will allow to get the input

// nextline will accept the white spaces although next() will not accept any

Spaces.

String input = sc.nextLine();

System.out.println("Output : " + input);

**Java Type Casting**  
Type casting is when you assign a value of one primitive data type to another type. In Java, there are two types of casting:

* **Widening Casting (automatically)** - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double
* **Narrowing Casting (manually)** - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte

// Widening Casting

// Widening casting is done automatically when passing a smaller size type to a larger size type:

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt); // Outputs 9

System.out.println(myDouble); // Outputs 9.0

/\* Narrowing Casting  
Narrowing casting must be done manually by placing the type in parentheses in front of the value:\*/

double myDouble = 9.78;

int myInt = (int) myDouble; // Manual casting: double to int

System.out.println(myDouble); // Outputs 9.78

System.out.println(myInt); // Outputs 9

**Operators:**Operators are used to perform operations on variables and values. The value is called an operand while the operation is defined by an operator

Java Divide the operators into the following groups.

* Arithmetic Operators
* Assignment Operators
* Comparison operators
* Logical operators
* Bitwise Operators

Example of Comparison Operators

Scanner sc = new Scanner(System.in);

System.out.print("Enter Input: ");

String input = sc.nextLine();

int age = Integer.parseInt(input);

if (age >= 21) {

System.out.println("Congratz, You can drink alcohol");

}

else {

System.out.println("Opps, Sorry - No Alcohols For you!");

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Enter Input: 26

// Congratz, You can drink alcohol

Nested if /else and Comparison.

Scanner sc = new Scanner(System.in);

System.out.print("Enter Input: ");

String input = sc.nextLine();

int age = Integer.parseInt(input);

if (age >= 21) {

System.out.print("Enter Favorite Food: ");

String food = sc.nextLine();

if (food.equals("burger")) {

System.out.println("Hey, My favorite is Cheese Burger");

}

else {System.out.println("not Mine");}

}

else if (age >= 15) {

System.out.println("Congratz, You can drink alcohol");}

else {

System.out.println("Opps, Sorry - No Alcohol For you!");

}

**Array**  
An array is collection of set size. Array are used to store multiple values in a single variable instead of declaring separate variables for each value. It will set all the values by default to null so they will not have any value but it exist.

// Create an Array

String[] newArr = new String[5];

newArr[0] = "Muhammad Shakir";

newArr[1] = "Bachelors in Computer Information System";

newArr[2] = "Web Development";

newArr[3] = "BBVA Compass";

newArr[4] = "Java Developer";

String retrieveValue = newArr[1];

System.out.println(retrieveValue);

// Output: Bachelors in Computer Information System

int[] nums = {11,43,55,23,51,221,532,123};

int numvalue = nums[6];

System.out.println("Index value of 6: " + numvalue);

// Output: 532

**For Loop**  
When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop

for (int i = 2; i <= 10; i+=2) {

System.out.println(i);

}

Array with For Loop

int array[] = {22,44,55,44,22,11,43};

for (int i= 0; i < array.length ; i++) {

if (array[i] == 22) {

System.out.println("Value " + array[i] + " Found at index: " + i);

}

}

// Value 22 Found at index: 0

// Value 22 Found at index: 4

// For Each Loop

String names[] = new String[4];

int array[] = {22,44,55,44,22,11,43};

int counter = 0;

for (int element: array) {

System.out.println("Counter "+ counter + " : " + element);

counter++;

}

Example of Both For Loop

String[] names = new String[3];

Scanner sc = new Scanner(System.in);

for (int i = 0; i < names.length ; i ++){

System.out.println("Enter the Input: ");

String input = sc.nextLine();

// Add input into an array

names[i] = input;

}

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

int counter = 0;

for (String num: names) {

System.out.println("Value : " + counter + " == " + num);

counter++;

}

**While Loop with Counter**

While Loop loops through a block of code as long as a specified condition is true.

Scanner sc = new Scanner(System.in);

System.out.print("Enter a Number: ");

int x = sc.nextInt(); int count = 1;

while(x!=10) {

System.out.print("Enter a Number again: ");

x = sc.nextInt();

count++;}

System.out.println("You have guessed the number in " + count + " times");

Set and Lists

// Sets and lists are from something known as the collection interface from java

/\*

List is a type of ordered collection that maintains the elements in insertion order

while Set is a type of unordered collection (Unique) so elements are not maintained any order. List allows duplicates while Set doesn't allow duplicate elements .

\*/

// Tree Set is one of the most important implementations of the Sorted Set interface in Java that uses a Tree for storage. The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided. This must be consistent with equals if it is to correctly implement the Set interface.

// HashSet standard implemention of sets

Set<Integer> value = new HashSet<Integer>();

value.add(1); value.add(2);

value.add(3); value.add(4);

value.add(1);

System.out.println("Set: " + value);

value.size();

int sizeSet = value.size();

System.out.println(sizeSet);

boolean find = value.contains(3);

System.out.println("Value is : " + find);

value.remove(3);

System.out.println("Set: " + value);

value.clear();

System.out.println("Set: " + value);

value.isEmpty();

System.out.println("Set: " + value);

**Java ArrayList**

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want. The syntax is also slightly different:

It has so many useful methods such add, get, set, sublist, remove, size and many more.

ArrayList<Integer> ms = new ArrayList<Integer>();

// Add the value in the array list

ms.add(93); ms.add(63); ms.add(26);

ms.add(93); ms.add(63); ms.add(26);

// Get the value from the array

ms.get(1);

// Set the value in the ArrayList

ms.set(1, 53);

ms.subList(1, 4);

System.out.println("Values in the ArrayList: " + ms);

System.out.println("Get the value from the ArrayList: " + ms.get(1));

System.out.println("Get the value from to from : " + ms.subList(1, 5));

// Values in the ArrayList: [93, 53, 26, 93, 63, 26]

// Get the value from the ArrayList: 53

// Get the value from to from : [53, 26, 93, 63]

System.out.println("Loop through the ArrayList:");

for (int i = 0; i < ms.size(); i++) {

System.out.println(ms.get(i));

}

Collections.sort(ms);

for (int i : ms) {

System.out.println(i);

}

// Sort the Array List

**HashMap:**

A HashMap store items in key/value pairs and it can access by an index of another type

Map ms = new HashMap();

// Key / value

ms.put("Name", "Muhammad Shakir");

ms.put("GPA", 3.88);

ms.put("Age", 2019-1993);

// Get the Value

System.out.println(ms.get("Name"));

System.out.println(ms.get(3.88));

System.out.println("All the values" + ms.values());

// Return Null but will not crash the program

// Print the entire Map

System.out.println(ms);

// HashMap is Very fast because it does not track the order.

Map ab = new TreeMap();

// It will print in the Sorted Order.

ab.put("Muhammad", "Shakir");

ab.put("Sumair", "Munawar");

ab.put("Abdul", "Basit");

System.out.println(ab);

Map sm = new LinkedHashMap();

// It will print in the same order as adding the value in the Map

sm.put("Muhammad", "Shakir"); sm.put("Sumair", "Munawar");

sm.put("Abdul", "Basit"); System.out.println(sm);

// Exercise to count the letters from the String and store each letter in the map.

Map q = new HashMap();

String str = "Hello, my name is Muhammad Shakir. I just completed my bachlor's degree in Computer Information System from Jacksonville State University.";

for (char val: str.toCharArray()) {

if (q.containsKey(val)){

int old = (int) q.get(val);

q.put(val, old+1); }

else {

q.put(val, 1);

}

} // To remove from the map ms.remove(“a”);

System.out.println(q);

Sort the Array

int[] sortx = {-99, -9, 0, 1,2,3,5,11, 98};

Arrays.sort(sortx);

for (int i : sortx) {

{

System.out.print(i + " , ");

}

}

**Create a Static method and run it in the main function.**

public class OnlineLearning {

public static void main(String[] args) {

ms("Muhammad Shakir", 5);

}

public static void ms(String str, int x) {

for (int i = 0 ; i < x; i ++) {

System.out.println(i + " ==== " +str);

}

}

}