KOTLIN CHEATSHEET

LEARN KOTLIN FROM SCRATCH

Print a statement

Java

```java
System.out.print("Ayusch jain");
System.out.println("Ayusch jain");
```

Kotlin

```kotlin
print("Ayusch jain")
println("Ayusch jain")
```

Constants and Variables

Java

```java
String name = "Ayusch jain";
final String name = "Ayusch jain";
```

Kotlin

```kotlin
var name = "Ayusch jain"
val name = "Ayusch jain"
```

Assigning the null value

Java

```java
String otherName;
otherName = null;
```

Kotlin

```kotlin
var otherName : String?
otherName = null
```

Verify if value is null

Java

```java
if (text != null) {
  int length = text.length();
}
```

Kotlin

```kotlin
text?.let {
    val length = text.length
}
// or simply
val length = text?.length
```

Concatenation of strings

Java

```java
String firstName = "Ayusch";
String lastName = "Jain";
String message = "My name is: " + firstName + " " + lastName;
```

Kotlin

```kotlin
val firstName = "Ayusch"
val lastName = "Jain"
val message = "My name is: $firstName $lastName"
```

New line in string

Java

```java
String text = "First Line\n" +
              "Second Line\n" +
              "Third Line";
```

Kotlin

```kotlin
val text = """
        |First Line
        |Second Line
        |Third Line
        """.trimMargin()
```

Ternary Operations

Java

```java
String text = x > 5 ? "x > 5" : "x <= 5";

String message = null;
log(message != null ? message : "");
```

Kotlin

```kotlin
val text = if (x > 5)
              "x > 5"
           else "x <= 5"

val message: String? = null
log(message ?: "")
```

Bitwise Operators

Java

```java
final int andResult  = a & b;
final int orResult   = a | b;
final int xorResult  = a ^ b;
final int rightShift = a >> 2;
final int leftShift  = a << 2;
final int unsignedRightShift = a >>> 2;
```

Kotlin

```kotlin
val andResult  = a and b
val orResult   = a or b
val xorResult  = a xor b
val rightShift = a shr 2
val leftShift  = a shl 2
val unsignedRightShift = a ushr 2
```

Check the type and casting

Java

```java
if (object instanceof Car) {
}
Car car = (Car) object;
```

Kotlin

```kotlin
if (object is Car) {
}
var car = object as Car

// if object is null
var car = object as? Car // var car = object as Car?
```

Check the type and casting (implicit)

Java

```java
if (object instanceof Car) {
   Car car = (Car) object;
}
```

Kotlin

```kotlin
if (object is Car) {
   var car = object // smart casting
}

// if object is null
if (object is Car?) {
   var car = object // smart casting, car will be null
}
```

Multiple conditions

Java

```java
if (score >= 0 && score <= 300) { }
```

Kotlin

```kotlin
if (score in 0..300) { }
```

Multiple Conditions (Switch case)

Java

```java
int score = // some score;
String grade;
switch (score) {
        case 10:
        case 9:
                grade = "Excellent";
                break;
        case 8:
        case 7:
        case 6:
                grade = "Good";
                break;
        case 5:
        case 4:
                grade = "OK";
                break;
        case 3:
        case 2:
        case 1:
                grade = "Fail";
                break;
        default:
            grade = "Fail";
}
```

Kotlin

```kotlin
var score = // some score
var grade = when (score) {
        9, 10 -> "Excellent"
        in 6..8 -> "Good"
        4, 5 -> "OK"
        in 1..3 -> "Fail"
        else -> "Fail"
}
```

For-loops

Java

```
for (int i = 1; i <= 10 ; i++) { }

for (int i = 1; i < 10 ; i++) { }

for (int i = 10; i >= 0 ; i--) { }

for (int i = 1; i <= 10 ; i+=2) { }

for (int i = 10; i >= 0 ; i-=2) { }

for (String item : collection) { }

for (Map.Entry<String, String> entry: map.entrySet()) { }
```

Kotlin

```
for (i in 1..10) { }

for (i in 1 until 10) { }

for (i in 10 downTo 0) { }

for (i in 1..10 step 2) { }

for (i in 10 downTo 0 step 2) { }

for (item in collection) { }

for ((key, value) in map) { }
```

Collections

Java

```
final List<Integer> listOfNumber = Arrays.asList(1, 2, 3, 4);

final Map<Integer, String> keyValue = new HashMap<Integer, String>();
map.put(1, "Ayusch");
map.put(2, "Anuj");
map.put(3, "AndroidVille");

// Java 9
```

```java
final List<Integer> listOfNumber = List.of(1, 2, 3, 4);

final Map<Integer, String> keyValue = Map.of(1, "Ayusch",
                                             2, "Anuj",
                                             3, "AndroidVille");
```

Kotlin

```kotlin
val listOfNumber = listOf(1, 2, 3, 4)
val keyValue = mapOf(1 to "Ayusch",
                     2 to "Anuj",
                     3 to "AndroidVille")
```

---

for each

Java

```java
// Java 7 and below
for (Car car : cars) {
  System.out.println(car.speed);
}

// Java 8+
cars.forEach(car -> System.out.println(car.speed));

// Java 7 and below
for (Car car : cars) {
  if (car.speed > 100) {
    System.out.println(car.speed);
  }
}

// Java 8+
cars.stream().filter(car -> car.speed > 100).forEach(car ->
System.out.println(car.speed));
cars.parallelStream().filter(car -> car.speed > 100).forEach(car ->
System.out.println(car.speed));
```

Kotlin

```kotlin
cars.forEach {
    println(it.speed)
}

cars.filter { it.speed > 100 }
      .forEach { println(it.speed)}

// kotlin 1.1+
cars.stream().filter { it.speed > 100 }.forEach { println(it.speed)}
cars.parallelStream().filter { it.speed > 100 }.forEach { println(it.speed)}
```

Splitting arrays

java

```java
String[] splits = "param=car".split("=");
String param = splits[0];
String value = splits[1];
```

kotlin

```kotlin
val (param, value) = "param=car".split("=")
```

Defining methods

Java

```java
void doSomething() {
   // logic here
}
```

Kotlin

```kotlin
fun doSomething() {
   // logic here
}
```

Variable number of arguments

Java

```java
void doSomething(int... numbers) {
   // logic here
}
```

Kotlin

```kotlin
fun doSomething(vararg numbers: Int) {
   // logic here
}
```

Defining methods with return

Java

```java
int getScore() {
   // logic here
   return score;
}
```

Kotlin

```kotlin
fun getScore(): Int {
   // logic here
   return score
}

// as a single-expression function

fun getScore(): Int = score

// even simpler (type will be determined automatically)

fun getScore() = score // return-type is Int
```

Returning result of an operation

Java

```java
int getScore(int value) {
    // logic here
    return 2 * value;
}
```

Kotlin

```kotlin
fun getScore(value: Int): Int {
   // logic here
   return 2 * value
}

// as a single-expression function
fun getScore(value: Int): Int = 2 * value

// even simpler (type will be determined automatically)

fun getScore(value: Int) = 2 * value // return-type is int
```

Constructors

Java

```
public class Utils {

    private Utils() {
      // This utility class is not publicly instantiable
    }

    public static int getScore(int value) {
        return 2 * value;
    }

}
```

Kotlin

```
class Utils private constructor() {

    companion object {

        fun getScore(value: Int): Int {
            return 2 * value
        }

    }
}

// another way

object Utils {

    fun getScore(value: Int): Int {
        return 2 * value
    }

}
```

Getters and Setters

Java

```
public class Developer {

    private String name;
    private int age;

    public Developer(String name, int age) {
```

```java
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Developer developer = (Developer) o;

        if (age != developer.age) return false;
        return name != null ? name.equals(developer.name) : developer.name == null;

    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }

    @Override
    public String toString() {
        return "Developer{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

Kotlin

```
data class Developer(var name: String, var age: Int)
```

Cloning or copying

Java

```
public class Developer implements Cloneable {

    private String name;
    private int age;

    public Developer(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return (Developer)super.clone();
    }
}

// cloning or copying
Developer dev = new Developer("Ayusch", 30);
try {
    Developer dev2 = (Developer) dev.clone();
} catch (CloneNotSupportedException e) {
    // handle exception
}
```

Kotlin

```
data class Developer(var name: String, var age: Int)

// cloning or copying
val dev = Developer("Ayusch", 30)
val dev2 = dev.copy()
// in case you only want to copy selected properties
val dev2 = dev.copy(age = 25)
```

Class methods

Java

```
public class Utils {

    private Utils() {
      // This utility class is not publicly instantiable
    }

    public static int triple(int value) {
        return 3 * value;
    }

}

int result = Utils.triple(3);
```

Kotlin

```
fun Int.triple(): Int {
  return this * 3
}

var result = 3.triple()
```

Defining uninitialized objects

Java

```
Person person;
```

Kotlin

```
internal lateinit var person: Person
```

---

enum

Java

```
public enum Direction {
        NORTH(1),
        SOUTH(2),
        WEST(3),
        EAST(4);

        int direction;

        Direction(int direction) {
            this.direction = direction;
        }

        public int getDirection() {
            return direction;
        }
    }
```

Kotlin

```
enum class Direction constructor(direction: Int) {
    NORTH(1),
    SOUTH(2),
    WEST(3),
    EAST(4);

    var direction: Int = 0
        private set

    init {
        this.direction = direction
    }
}
```

Sorting List

Java

```
List<Profile> profiles = loadProfiles(context);
Collections.sort(profiles, new Comparator<Profile>() {
    @Override
    public int compare(Profile profile1, Profile profile2) {
        if (profile1.getAge() > profile2.getAge()) return 1;
        if (profile1.getAge() < profile2.getAge()) return -1;
        return 0;
    }
});
```

Kotlin

```
val profile = loadProfiles(context)
profile.sortedWith(Comparator({ profile1, profile2 ->
    if (profile1.age > profile2.age) return@Comparator 1
    if (profile1.age < profile2.age) return@Comparator -1
    return@Comparator 0
}))
```

Anonymous Class

Java

```java
 AsyncTask<Void, Void, Profile> task = new AsyncTask<Void, Void, Profile>() {
    @Override
    protected Profile doInBackground(Void... voids) {
        // fetch profile from API or DB
        return null;
    }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        // do something
    }
};
```

Kotlin

```kotlin
val task = object : AsyncTask<Void, Void, Profile>() {
    override fun doInBackground(vararg voids: Void): Profile? {
        // fetch profile from API or DB
        return null
    }

    override fun onPreExecute() {
        super.onPreExecute()
        // do something
    }
}
```