IEEE
University of Porto - Student Branch

IEEE computer society

NO BS GUIDE TO **JAVA**

IEEE UP Student Branch

PREREQUISITES

- Pretty basic programming knowledge.

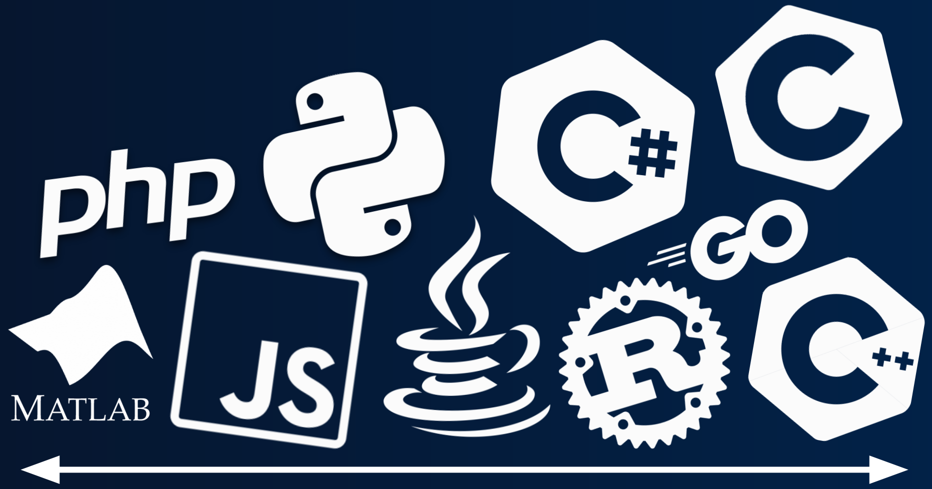
```
for (int i = 0; i < 5; i++) {
    print('If you don't understand this block of
    code you're likely f*cked.');
}
```

- A code editor or IDE of your choosing.
  I'll be using Visual Studio Code with Code Runner.

- *Any Java JDK version.*

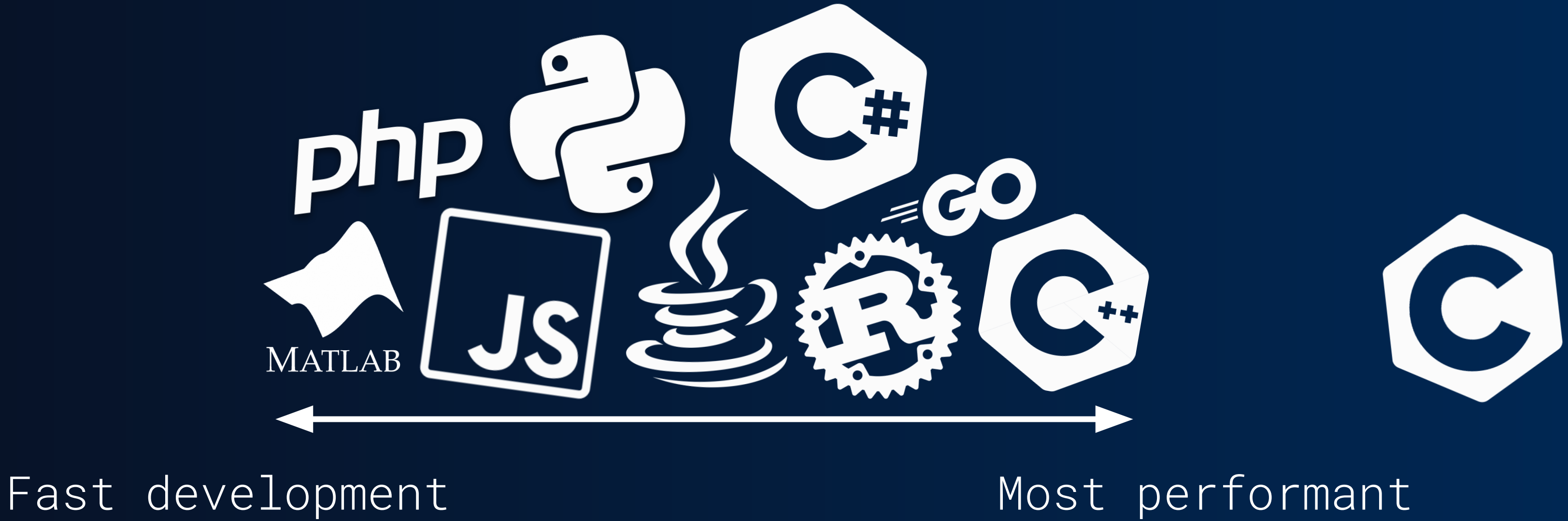
```java
public class Main {
    public static String reverser(String str) {
        StringBuilder reverse = new StringBuilder();
        for (int idx = str.length() - 1; idx >= 0; idx--) {
            reverse.append(str.charAt(idx));
        }
        return reverse.toString();
    }

    public static void main(String[] args) {
        String hello = “Hello world!”;
        System.out.println(reverser(hello));
    }
}
```

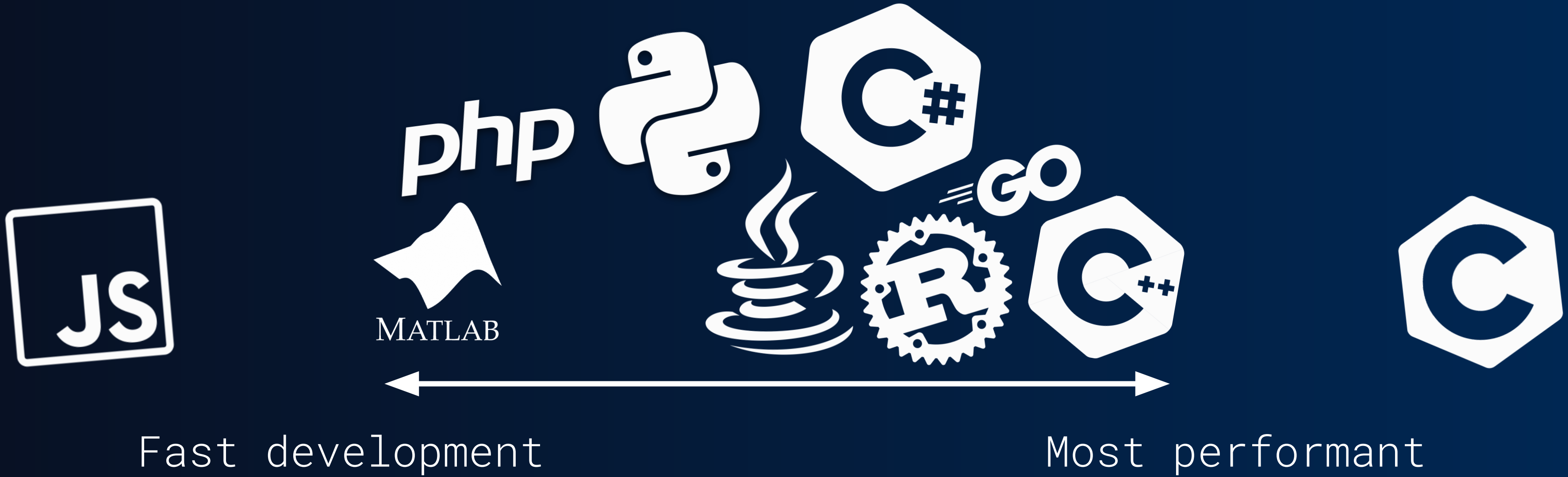
```
hello = ‘Hello world!’
print(hello[::-1])
```

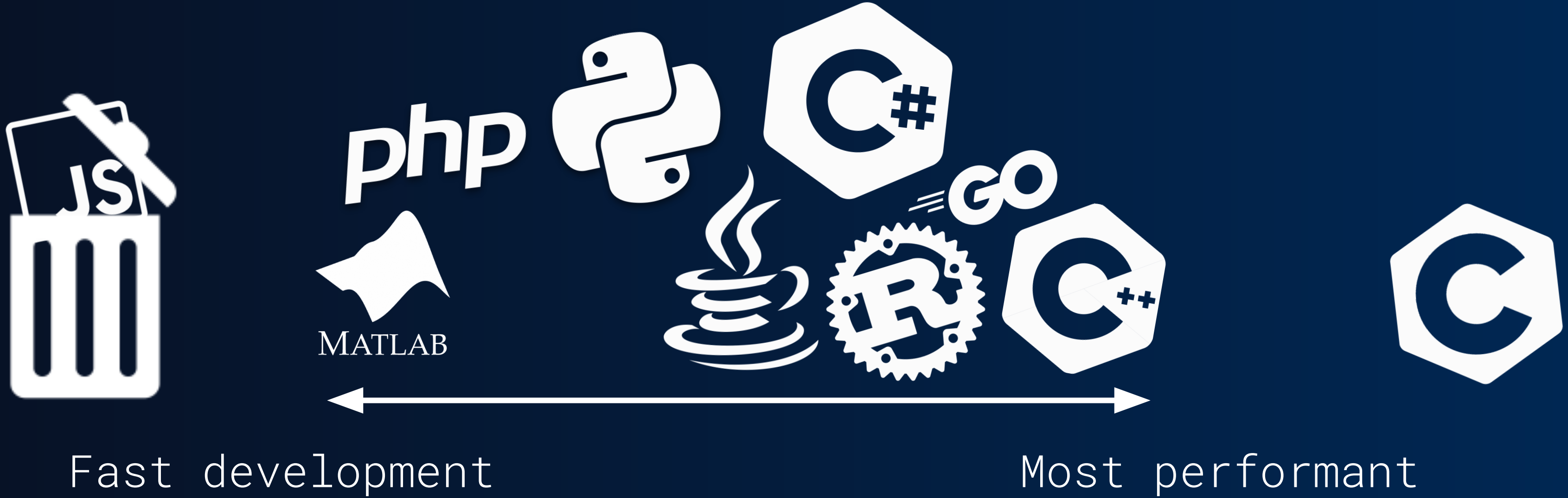


Fast development

Most performant


php
MATLAB
JS
GO
Fast development
Most performant


JS
php
GO
MATLAB
Fast development
Most performant


JS
php
MATLAB
GO
Fast development
Most performant

firstly, you write your source code

```
HelloWorld.java
```

```
javac HelloWorld.java
```

```
HelloWorld.class
```

```
java HelloWorld
```

```
Hello, World
```

```
HelloWorld.java
```

```
javac HelloWorld.java
```

compile the source, translating it to a lower-level language (e.g. Assembly)

```
HelloWorld.class
```

```
java HelloWorld
```

```
Hello, World
```

HelloWorld.java

```
javac HelloWorld.java
```

freshly generated bytecode (intermediate platform-independent binary code)

HelloWorld.class

```
java HelloWorld
```

```
Hello, World
```

```
HelloWorld.java
```

```
javac HelloWorld.java
```

```
HelloWorld.class
```

```
java HelloWorld
```

the Java interpreter launches a JVM (Java Virtual Machine) and executes main

```
Hello, World
```

```
HelloWorld.java
```

```
javac HelloWorld.java
```

```
HelloWorld.class
```

```
java HelloWorld
```

nice

```
Hello, World
```

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println(“Hello world”);
    }
}
```

other access
modifiers are
possible

only one class
per file

therefore the file name
must be HelloWorld.java

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println(“Hello world”);
    }
}
```

main returns nothing

the entry point is always the main function

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println(“Hello world”);
    }
}
```

it’s static because of a lot of reasons I’ll explain to you later on

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello world");
    }
}
```

the standard output stream (fancy words for console)

print in a new line

the string that should be printed

int	int[]
float	String
double	Integer
char	Boolean
boolean	

Garbage collection

Generate a Fibonacci series

0 1 1 2 3 5 8 13

I understand this is barely a challenge, just making sure we're all in sync.

#1

Easy peasy lemon squeezy

Powerball: 1st Iteration

Generate and print 6 random numbers from 1 to 49

java.util.Random

#2

Write something funny here

Powerball: 2nd Iteration

Same thing, just avoid duplicates now

java.util.ArrayList

Write something funny here

Powerball: 3rd Iteration

Sort before printing

`Collections.sort(ArrayList)`

Write something funny here

Powerball: Final Iteration

Generate N (inserted by the user) random bets

System.in & Scanner

#2

Write something funny here

The year is 2300 and you're programming an application which stores info about every planet humans have colonized.

```
boolean earthHasWater = true
String[] earthSatellites = {“Moon”}
float earthAvgTemp = 14.9
```

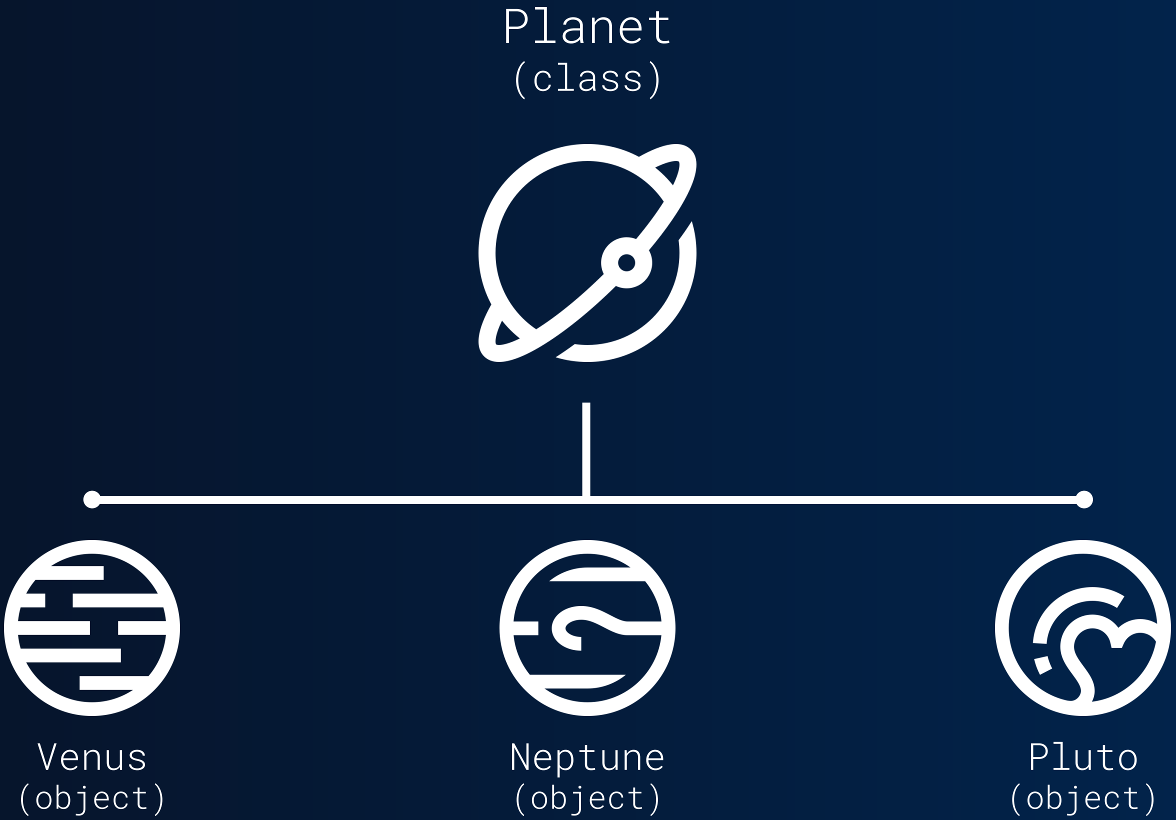
```
boolean earthHasWater = true
String[] earthSatellites = {“Moon”}
float earthAvgTemp = 14.9
```

```
boolean marsHasWater = false
String[] marsSatellites = {“Phobos”, “Deimos”}
float marsAvgTemp = -63
```

```
boolean earthHasWater = true
String[] earthSatellites = {“Moon”}
float earthAvgTemp = 14.9
```

```
boolean marsHasWater = false
String[] marsSatellites = {“Phobos”, “Deimos”}
float marsAvgTemp = -63
```

```
boolean jupiterHasWater = idk lol
String[] jupiterSatellites = way too many
float jupiterAvgTemp = a direct bite on ice cream
```


Planet
(class)
Venus
(object)
Neptune
(object)
Pluto
(object)

public protected default private

public protected default private

seen by every class
on any package

public

protected

hidden from
non-subclasses in
other packages

default

private

public protected default private

only seen by the
package's scope

public protected default private

hidden from everyone

github.com/ieeeupsb/**no-bs-guide-to-java**

1. Create a **Planet** class, containing a **name** and **average Kelvin temperature**.
2. Create a **Civilization** class with a **name** and a (mutable) **list of planets**. Add a method which allows **new planets to be pushed** into the list.
3. Add an attribute to the Planet class which **tracks whether is has been colonized or not**.
4. Create a method which returns the **coldest temperature for a planet** owned by a civilization.
5. Create a **Space** class with a **static method** which - given two civilizations - returns the one with the **highest number of planets colonized**.

- Implement **getter methods** for **all attributes**.
- Look at the **Test.java** code to better understand what we're looking for.

#3

Difficult difficult lemon difficult

Where to next?

Kotlin

Android Studio

Abstract classes
Interfaces
Inheritance
Java Swing/FX
JUnit