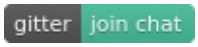
Table of Contents

কোর্স এর মুল পাতা | HowToCode মুল সাইট | সবার জন্য প্রোগ্রামিং ব্লগ | পিডিএফ ডাউনলোড

Scala দি নেক্সট বিগ থিং



সংক্ষেপ

স্ক্যালা- স্ক্যালেবল ল্যাংগুয়েজ।

আপনি যদি ইতিমধ্যে জাভা প্রোগ্রামার হয়ে থাকেন, আপানকে সু-স্বাগতম। আশাকরি আপনাকে স্ক্যালা প্রোগ্রামিং এর সমুদ্রে পা ভেজানো চক্রান্তে আমি খানিকটা সফল। আর যদি না হয়ে থাকেন, তাহলে কোন কথাই নেই, সুপার-ডুপার ওয়েলকাম।

> ### Statutory warning
>
> **This book may contain unexpected misspellings. Reader Feedback Requested.**

উপক্রমণিকা

আমি নিশ্চিত যে মোটামুটি ভাবে সবাই রড জনসন(Rod Johnson) কে চেনে। তাকে না চেনা খুব বড় অন্যায়, এখনি গুগুল করে জেনে নিন। তবে রড জনসনকে না চিনলেও স্প্রিং ফ্রেমওয়ার্ক এর নাম শুনেনি এমন লোক খুব কম পাওয়া যাবে।

একটা সময় ছিল যখন জাভা-এন্টারপ্রাইজ এপ্লিকেশান লেখা অনেক বেশি ক্লান্তিকর ছিল, তখন প্রথম আলোর মুখ দেখায় রড জনসন তার "Expert One-on-One J2EE Development Without EJB" বইটি লিখে। তারপর পরের ইতিহাস সম্পূর্ণ ভিন্ন। জুন ২০০৩ এ স্প্রিং ফ্রেমওয়ার্ক এর প্রথম রিলিজ হয় এবং এর পর সারা পৃথিবীর মানুষ বিপুল উৎসাহে এই ফ্রেমওয়ার্ক ব্যবহার করে আসছে। JVM ওয়েব ফ্রেমওয়ার্ক গুলোর মধ্যে স্প্রিং সবার উপরে । রড জনসন হচ্ছে সেই স্প্রিং ফ্রেমওয়ার্ক এর জনক। VMWare ২০০৯ সালে SpringSource কিনে নেওয়া পর্যন্ত সেখানে সিইও হিসেবে দায়িত্ব পালন করে এসেছেন। তিনি বর্তমানে **Typesafe, Inc** এর বোর্ড অব ডিরেক্টর দের মধ্যে একজন।

যাহোক মূল কথায় ফিরে আসি,

ScalaDays 2013 Kaynote এ রড জনজন একটা বোল্ড মন্তব্য করেন। সেটি হলো -

> I believe in 2018 Scala will be the leading newer language. I believe its currently breaking out of pack of cycled second-tier languages and it will be the biggest things since Java.

আরো কিছু মন্তব্য -

> If I were to pick a language today other than Java, it would be Scala.
>
> -James Gosling(father of Java)
>
> I can honestly say if someone had shown me the Programming Scala book by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy.
>
> -James Strachan (creator of Groovy)

সুতরাং বুঝা যাচ্ছে, স্ক্যালা ইজ দি বিগ থিং। তো শুরু করা যাক।

প্রোগ্রামিং এর দুনিয়ায় মোটামুটিভাবে আমরা দুই ধরণের প্রোগ্রামিং সাথে বেশি পরিচিত-

১. ফাংশনাল প্রোগ্রামিং ২. অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং

দুটিরই ব্যপকতা অনেক। এদের মধ্যে মূল পার্থক্য হচ্ছে, ফাংশনার প্রোগ্রামিং এ ডাটা এক জায়গা থাকে, আর মেথড/ফাংশন গুলো অন্য কোথায় থাকে। আমরা ডাটা একসেস করি প্যাটার্ন ম্যাচিং এর মাধ্যমে। আর অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং মডেল বলে, এটি সম্পূর্ণ ভুল, এটা মোটেও করা যাবে না। মেথড/ফাংশন গুলো সেখানেই রাখতে হবে যেখানে ডাটা রাখা আছে। এইভাবে দুটি কমিউনিটি আলাদা হয়ে যায়।

তবে এটা বলতে কারো আপত্তি নেই যে, দুটি প্রোগ্রামিং মডেল এর স্বপক্ষে ও বিপক্ষে যুক্তি আছে। মজার ব্যপার হলো স্ক্যালা এমন একটি ল্যাংগুয়েজ যা এই দুটি কমিউনিটিকে এক করে দিতে সক্ষম হয়েছে। স্ক্যালা একটি পিউর অবজেক্ট ওরিয়েন্টেড ল্যাংগুয়েজ এবং সেই সাথে ফাংশনাল ল্যাংগুয়েজ-ও। এটি কিভাবে সম্ভব সেটি নিয়ে অনেকেরই সন্দেহ থাকতে পারে, তবে ঘটনা সত্যি।

কিন্তু কেন স্ক্যালা শিখতে যাবো সেটা নিয়েও সংশয় থাকতেই পারে। আজকের এই লেখার মূল উদ্দেশ্য অবশ্য সেটাই। কেন স্ক্যালা এর উত্তর দেওয়া।

সংক্ষেপে দীর্ঘ গল্প

স্ক্যালা অপেক্ষাকৃত নতুন প্রোগ্রামিং ল্যাংগুয়েজ, মাত্র ১০ বছর হয়েছে। এটি স্ট্যাটিক্যালী টাইপড এবং অবজেক্ট ওরিয়েন্টেড ও ফাংশনাল প্রোগ্রামিং ল্যাংগুয়েজ এর হাইব্রিড যা কিনা জাভা ভার্চুয়াল মেশিনে চলে (You know JMV languages are always hot (!))। খুব দ্রুত এটি জনপ্রিয় হয়ে যাচ্ছে (its getting hotter day by day)। এর বড় একটা কারণ হতে পারে কারণ এটি জাভার সব থেকে ভাল বিকল্প।

কেন ??

১. স্ট্যাটিক্যালি টাইপড

আমরা সাধারণত দুই ধরণের টাইপ সিস্টেম নিয়ে কথা বলে থাকি- স্ট্যাটিক্যালি টাইপড এবং ডাইনামিক্যালি টাইপড। দুই ধরণের ল্যাংগুয়েজের প্রয়োজনীয়তা দুই রকম। স্ট্যাটিক টাইপড ল্যাংগুয়েজ প্রত্যেকটি ভ্যারিয়্যবল আগে টাইপ ইনফরমেশন থাকতে হয়, সেটি টাইপ এবং অবজেক্ট দুটির ক্ষেত্রেই সত্য। উদাহরণ- Java-

```
String myName = "Bazlur Rahman";
```

কিন্তু ডাইনামিক টাইপড ল্যাংগুয়েজ এর ক্ষেত্রে এর দরকার হয় না। যেমন – python -

```
myName = "Bazlur Rahman"
```

স্ট্যাটিক্যালি টাইপড সিস্টেমে কম্পাইলার কম্পাইল করার সময় সবকিছু চেক করে থাকে। যদি কোন ধরণের ভুল থাকে তাহলে সেগুলো কম্পাইল করার সময় ধরা পরে। অর্থাৎ কেও যদি ফ্লোটিং পয়েন্ট ভ্যারিয়বল এ স্ট্রিং এসাইন করে ফেলে ভুল করে, তাহলে সেটি কম্পাইল করার সময়-ই ধরা পরবে, কিন্তু ডাইনামিক টাইপ সিস্টেম এ তা কম্পাইল টাইমে ধরা না পরে রান টাইম এ ধরা পরবে।

সহজ কথায়- স্ট্যাটিক টাইপ সিস্টেম ভাল কারণ এটি আপনাকে ঝামেলা থেকে দূরে রাখবে । আর ডাইনামি টাইপ সিস্টেম ভাল এটি আপানাকে আপনার মতো কাজ করতে দেবে এবং আপনি দ্রুত কাজ করতে পারবেন। সুতরাং আপনি নিজেই ঠিক করে নিন কোনটা আপনার জন্যে ভাল।

ব্যক্তিগত ভাবে আমি টাইপসেইফ প্রোগ্রামিং ল্যাংগুয়েজ এর গ্রেট ফ্যান। কারণ- কম্পাইলার কম্পাইল করার সময় সব ধরণে ধরনের বৈধতা যাচাই করে, এবং আপনি একটি ভ্যারিয়বলে ভুল টাইপ নির্ধারণ করার চেষ্টা করলেই আপনাকে সাথে সাথে জানিয়ে দেবে। যেহেতু আমি **Linus Torlvads** নই, সুতরাং আমি ভুল করতেই পারি।

স্ট্যাটিক টাইপ সিস্টেম-এ অনেক সহজে এবং খুব বেশি চিন্তা ভাবনা না করেই কোড রিফ্যাক্টরিং করা যায়। ধরা যাক, একটা মেথড ইন্টিজার প্যারিমিটার নেই। আমি সেটা রিফ্যাক্টর করে স্ট্রিং করতে চাই। এবং সেটি করতে গিয়ে অনেক যায়গায় পরিবর্তন করতে হতে পারে। স্ট্যাটিক টাইপ সিস্টেম এ কোড বেইজ শুধুমাত্র আবার রি-কম্পাইল করার সাথে সাথেই জেনে যাবো আর কোথায় কোথায় পরিবর্তন করতে হবে। মোট কথা - স্ট্যাটিক টাইপ আমাকে আমার পরিবর্তিত কোড যে ঠিক আগের মতই কাজ করবে তার যথেষ্ট নিশ্চয়তা প্রদান করে।

এ দিক থেকে স্ক্যালার টাইপ সিস্টেম অনেক বেশি উন্নত।

Interoperability

স্ক্যালা JVM ল্যাংগুয়েজ। এটি এমন ভাবে তৈরি করা হয়েছে যাতে করে Java-র সাথে সিমলেসলি কাজ করে পারে। স্ক্যালা সরাসরি Java-র ক্লাস, মেথড, ফিল্ডস কল করতে পারে এবং জাভা ক্লাস ইনহেরিট করতে পারে এমনকি জাভার ইন্টারফেইস ইম্প্লিমেন্ট করতে পারে। সুতরাং জাভার সমস্ত লাইব্রেরি স্ক্যালা ব্যবহার করতে পারে। এর থেকে বেশি Cool আর কি হতে পারে ।

সংক্ষিপ্ত

স্ক্যালা কোড সাধারণত খুব ছোট হয়। টিপিক্যালি একটি জাভা কোডকে স্ক্যালাতে লিখলে কোড অব লাইন অর্ধেক হয়ে যায়। উদাহরণ-

```
public class MyClass {
    private int index;
    private String name;

    public MyClass(int index, String name) {
        this.index = index;
        this.name = name;
    }
}
```

এই সেইম কোড স্ক্যালাতে হবে এক লাইন -

```
  class MyClass(index: Int, name: String)
```

এই কোড হুবহু উপরের জাভা কোডটির মতই কাজ করে ।

ধরা যাক, আমি একটা স্ট্রিং ভ্যারিয়েবল এ আপার কেইস লেটার আছে কিনা সেটা বের করতে চাই-

আমরা যদি জাভাতে করতে চাই তাহলে -

```
boolean hasUpperCaseLetter = false;

for (int i = 0; i < name.length(); i++) {
    if (Character.isUpperCase(name.charAt(i))) {
        hasUpperCaseLetter = true;
        break;
    }
}
```

কিন্তু সেই একই কোড স্ক্যালা তে করতে পারি এক লাইন এ -

```
val hasUpperCaseLetter = name.exists(_.isUpper)
```

Good parts from other languages

স্ক্যালা মূলত অনেক গুলো ল্যাংগুয়েজ এর গুড পার্ট গুলো নিয়ে এবং অনেক দিনের গবেষণার ফসল। ইনোভেশান এর দিক থেকে স্ক্যালাতে খুব বেশি নতুন কিছু নেই, এর ফিচার গুলো অন্য কোন কোন ল্যাংগুয়েজ এ রয়েছে। কিন্তু স্ক্যালা এর উয়িন অন্য যায়। এটি মূলত সবগুলো গুড পার্ট এক যায়গায় সমন্বিত করা।

সিনট্যাক্স এর দিক থেকে স্ক্যালার বড় অংশ জাভা এবং সি শার্প থেকে ধার করা। এর মানে এর এটি সি এবং সি++ থেকেও ধার করেছে। রুবি এবং স্মলটক থেকে এর অবেজেক্ট মডেল ধার করা। মোট কথা- এটিতে অনেক গুলো ল্যাংগুয়েজ থেকে ভাল ভাল পার্ট গুলো নিয়ে একযায়গা সমন্বিত করা হয়েছে - যেমন - Algol, Simula, SML, Ocam, F#, ML, Haskell, Erlang, Java, C# etc.

Type Inference

স্ক্যালা তে চমৎকার টাইপ ইনফারেন্স সিস্টেম থাকায় অনেক ক্ষেত্রেই টাইপ ইনফরমেশান লিখতে হয় না। যেমন -

```
val x = 1 + 2  // type Integer
val y = x.toString // string type

def increment(x: Int) = x + 1 //  method returns integer
```

ওপেন সোর্স

এই বইটি মূলত স্বেচ্ছাশ্রমে লেখা এবং বইটি সম্পূর্ন ওপেন সোর্স । এখানে তাই আপনিও অবদান রাখতে পারেন লেখক হিসেবে । আপনার কন্ট্রিবিউশান গৃহীত হলে অবদানকারীদের তালিকায় আপনার নাম যোগ করে দেওয়া হবে ।

এটি মূলত একটি গিটহাব রিপোজিটোরি যেখানে এই বইয়ের আর্টিকেল গুলো মার্কডাউন ফরম্যাটে লেখা হচ্ছে । রিপোজটরিটি ফর্ক করে পুল রিকুয়েস্ট পাঠানোর মাধ্যমে আপনারাও অবদান রাখতে পারেন ।

হাতে খড়ি

চলুন তাহলে স্ক্যালা-তে কোড করা শুরু করা যাক। প্রথমত আমরা স্ক্যালা ইন্টাপ্রেটার দিয়ে শুরু করবো। আমরা দেখবো স্ক্যালা ইন্টারপ্রেটার কিভাবে ইন্ডাস্ট্রি-স্ট্যান্ডার্ড পকেট ক্যালকুলেটর এর মতো কাজ করে।

এর জন্যে প্রথমত আমাদের স্ক্যালা ইনভাইরনমেন্ট সেটাপ করতে হবে। এর জন্যে আমাদের যা যা লাগবে তা হলো-

- Java – Java Development Kit ইনস্টলেশান
- স্ক্যালা ইনস্টলেশান

জাভা ইনস্টল করতে হলে আমাদের প্রথমে জাভা ডাউনলোড করতে হবে।

আমি ঠিক করেছি এই পুরো টিউটোরিয়াল গুলো লিনাক্স মেশিন ব্যবহার করে করবো (এর পেছনের কারণ, আমার মেশিনে একমাত্র ওএস লিনাক্স)। সুতরাং আপনার মেশিনে যদি লিনাক্স থাকে তাহলে খুবই চমৎকার, না থাকলে লিনাক্স ইনস্টল করে নেওয়া ভাল আইডিয়া, অথবা ভার্চুয়াল বক্স দিয়েও কাজ চলে যাবে।

জাভা ইনস্টলেশান-

জাভা ইনস্টল করতে নিচের ধাপ গুলো apply করতে হবে-

ধাপ ১: নিচের লিংক থেকে জাভা ডাউনলোড করে নিন।

Oracle JDK 7 Download Link

ধাপ ২: এরপর টার্মিনাল থেকে যেখানে জাভা ডাউনলোড হয়েছে সেখানে যান-

```
cd ~/Download
```

ধাপ ৩: এবার JDK ইনস্টল করি-

```
sudo tar -xzvf  jdk-7u21-linux-i586.tar.gz --directory=/usr/local/

sudo ln -s /usr/local/[jdk_folder_name]/ /usr/local/jdk
```

jdk_folder_name - আপনার পছন্দমত একটি নাম দিন।

ধাপ ৪: আবার টার্মিনালে ফিরে যান- .bashrc অপেন করুন।

```
sudo gedit .bashrc
```

ধাপ ৫ : .bashrc ফাইল-এ নিচের লাইনটি এড করুন।

```
export JAVA_HOME=/usr/local/jdk
```

Save and close .bashrc file.

ধাপ ৬: কম্পাইল .bashrc ফাইল

```
source .bashrc
```

ধাপ ৭: এবার পরিক্ষা করে দেখা যাক জাভা ইনস্টল হয়েছে কিনা। আবার টার্মিনাল ওপেন করুন এবং নিচের লাইনটি টাইপ করুন।

```
java -version
```

যদি সবকিছু ঠিকঠাক থাকে তাহলে আপনি নিচের তথ্য গুলো দেখতে পারবেন-

```
java version "1.7.0_65"
Java(TM) SE Runtime Environment (build 1.7.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)
```

Scala installation

কমান্ড লাইন থেকে Scala চালানোর জন্য, নিচের লিংক থেকে বাইনারিটি ডাউনলোড করুন এবং আর্কাইভটি /usr/local/scala ফোল্ডার-এ আনপ্যাক করুন।

http://www.scala-lang.org/download/

এবার টার্মিনালে যান- .bashrc অপেন করুন।

```
    sudo gedit .bashrc
```

এবং নিচের লাইন গুলো .bashrc ফাইল এ এড করুন।

```
    export SCALA_HOME=/usr/local/scala-2.11.2/bin
    export PATH=$PATH:$SCALA_HOME
```

সবকিছু ঠিকঠাক থাকলে চলুন এবার টার্মিনালে ফিরে যাই।

টাইপ scala এবং প্রেস ইন্টার।

```
:~$ scala
Welcome to Scala version 2.11.2 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_65).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

এরপর লিখুন 8*9. আপনি সাথে সাথেই উত্তর পেয়ে যাবেন। উদাহরণ-

```
scala> 8*9
res0: Int = 72
```

একটি মজার বিষয় লক্ষ্য করুন, আপনার উত্তরটি res0 ভ্যারিয়বল এ স্টোর হয়েছে। আপনি চাইলে এটি অন্য যায়গায় ব্যবহার করতে পারবেন। উদাহরণ-

```
scala> res0 * 5
res1: Int = 360
```

চলুন অন্য কিছু চেষ্টা করে দেখি।

```
scala> "Hello, " + res1
res2: String = Hello, 360
```

আপনি লক্ষ্য করুন, ইন্টারপ্রেটারটি উত্তরটির সাথে এর টাইপ ইনফরমেশান-ও পদর্শন করে ।

আপনি চাইলে এখানে কোন মেথড ও কল করতে পারেন।

```
scala> res2.toUpperCase
res4: String = HELLO, 360

scala> res2.toCharArray
res5: Array[Char] = Array(H, e, l, l, o, ,,  , 3, 6, 0)

scala> res2.toLowerCase
res6: String = hello, 360
```

সুতরাং দেখা যাচ্ছে যে স্ক্যালা ইন্টারপ্রেটার প্রথমে একটি এক্সপ্রেশান পড়ে, তারপর এটিকে Evaluate করে , প্রিন্ট করে এবং এক্সপ্রেশান পড়ে। একে বলে - read-eval-print lool সংক্ষেপে REPL.

টেকনিক্যালি স্ক্যালা প্রোগ্রাম মোটেও ইন্টারপ্রেটার নয়। কারণ এর পেছনের দৃশ্য অন্যরকম। এটি দ্রুত আপনার ইনপুট কে কম্পাইল করে বাইট কোড রূপান্তরিত করে এবং যা জাভা ভার্চুয়াল মেশিন দ্বারা এক্সিকউট হয়। সুতরাং খুব দ্রুত কিছু করতে হলে REPL হচ্ছে আপনার চমৎকার বন্ধু।

Declaring variables

res0 , res1 পরিবর্তে আমরা আমাদের নিজেদের পছন্দমত ভ্যারিয়েবল ব্যবহার করতে পারি।

```
scala> val answer = 5* 99
answer: Int = 495
```

লক্ষ্য করুন, আপনাকে ভ্যারিয়বলের টাইপ বলে দিতে হচ্ছে না। স্ক্যালা নিজে নিজেই টাইপ বুঝে নিতে পারে। এটিকে বলে **Type Inference**। তবে আপনি চাইলে টাইপ বলে দিতে পারেন, এতে স্ক্যালা কোন আপত্তি করবে না।

```
scala> val hello:String = "Hello, world"
hello: String = Hello, world
```

আরো একটি বিষয় লক্ষ্য করুন, স্ক্যালাতে টাইপ সবসময় ভ্যারিয়বলের কিংবা ফাংশান এর পরে লিখতে হয়, যা কিনা অন্যান্য ল্যাংগুয়েজ থেকে আলাদা। আপনি যদি জাভা এবং স্ক্যালা একি সাথে ব্যবহার করেন তাহলে এটি একটু বিরক্তিকর হতে পারে কিন্তু এটি একটি চমৎকার উদ্দেশ্যে করা হয়েছে।

স্ক্যালাতে ভ্যারিয়্যবল ডিক্লেয়ার করার জন্যে দুই ধরণের Keyword ব্যবহার কর হয়- val এবং var।

যে ভ্যারিয়বল গুলো val ব্যবহার করে ডিক্ল্যায়ার করা হয় সেগুলো মূলত কনস্ট্যান্ট, এর কনটেন্ট আর পরিবর্তন করা যাবে না। কিন্তু যে সব ভ্যারিয়েবল এর কনটেন্ট পরিবর্তনীয় সেগুলোর ক্ষেত্রে var ব্যবহার করা হয়।

কোন ভ্যারিয়েবল যদি val ব্যবহার করে ডিক্ল্যায়ার করা হয় এবং আমরা তা পরিবর্তন করতে চাই তাহলে সাথে সাথে ইন্টারপ্রেটার কম্পাইলার ইরর দেখাবে।

```
scala> val x = 9
x: Int = 9

scala> x =6
<console>:8: error: reassignment to val
       x =6
         ^
```

আরেকটি মজার বিষয় লক্ষ্য করুন, আমরা কোন স্ট্যাটমেন্ট এর আগে সেমিকোলন ব্যবহার করি নি। স্ক্যালাতে সেমিকোলন বাধ্যতামূলক নয়। তবে যদি একি লাইন এ আমরা একাধিক স্ট্যাটমেন্ট ব্যবহার করি, তহালে সেমিকোল দিয়ে স্ট্যাটমেন্ট গুলোকে আলাদা করতে হবে।

ডাটা টাইপ

স্ক্যালাতে মূলত নিচের বিল্ট ইন ডাটাটাইপ গুলো ভ্যারিয়েবল হিসেবে ব্যবহার করা হয়।

```
Type |        Value Space
----| -------------------
Boolean|            true or false
Byte|        8 bit signed value
Short|        16 bit signed value
Char|        16 bit unsigned Unicode character
Int|        32 bit signed value
Long|        64 bit signed value
Float|        32 bit IEEE 754 single-precision float
Double|            64 bit IEEE 754 double-precision float
String|            A sequence of characters
```

এবং স্ক্যালাতে সবগুলো ডাটা টাইপ অবজেক্ট এবং এতে জাভা এর মতো কোন প্রিমিটিভ ডাটা টাইপ নেই।
এতে করে একটা মজার সুবিধে পাওয়া যাচ্ছে - আপনি সরা সরি এর মেথড গুলো কল করতে পারবেন। উদাহরণ-

```
scala> 1.toString
res7: String = 1

scala> 1.to(10)
res8: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

লক্ষ্য করুন, 1.toString লাইনে আমি পেরেন্থেসিস () ব্যবহার করি নি। কারণ স্ক্যালাতে এটি আবশ্যিক নয় ।

আমাদের যদি কোন Double নাম্বারকে Integer রূপান্তরিত করতে হয়, স্ক্যালাতে সি এর মতো কাস্টিং না করে আমরা toInt মেথড কল করা হয়।

```
scala> var a:Double = 9.6
a: Double = 9.6

scala> var b: Int = a.toInt
b: Int = 9
```

Arithmetic operation

স্ক্যালাতে এরিথমেটিক অপারেশন গুলো জাভা এবং সি++ এর মতই কাজ করে। যেমন-

```
val answer = 1 + 8 / 4
```

+ - / % & | ^ >> << এই অপারেটর গুলো মূলত তাদের ইউজ্যুয়াল কাজ গুলোই করে থাকে, তবে একটি অত্যন্ত মজার দৃষ্টিভঙ্গি আছে এখানে এবং সেটি হলো এগুলো মূলত মেথড। আপনি নিশ্চয় এটি শুনে আকাশ থেকে পরবেন। কিন্তু সত্যি হচ্ছে স্ক্যালা তে অন্যান্য ল্যাংগুয়েজ গুলোর মতো অর্থহীন কুসংস্কার নেই যে মেথড শুধুমাত্রই alphanumeric ই হতে হবে। আপনি যে কোন সিম্বল দিয়ে মেথড লিখতে পারেন।

```
    a +b
```

মূলত

```
a.+(b)
```

এর শর্টহ্যান্ড।

নিচের উদাহরণটি লক্ষ্য করুন

```
class Calculator {
  def doubleIt(a: Int) = a*a
}
```

এটি একটি ক্লাস এবং এতে একটি মেথড আছে। আমরা এই মেথডটি ব্যবহার করতে হলে আমাদের যা করতে হবে -

```
  var calculator = new Calculator()
  calculator.doubleIt(5)
```

উপরের মেথড কলটি আমরা চাইলে শর্টহ্যান্ড এ লিখতে পারি -

```
    calculator doubleIt 5
```

চমৎকার। প্রথম শব্দটি হলো ইনসট্যান্স এবং পরেরটি মেথড এবং এর পরেরটি হলো আর্গুমেন্ট।

ফাংশান

চলুন এবার দেখি কিভাবে স্ক্যালাতে ফাংশন লেখা যায়।

```
scala> def max(x:Int, y:Int): Int ={
     | if(x>y) x
     | else y
     | }
max: (x: Int, y: Int)Int

scala> max(9,5)
res9: Int = 9
```

লক্ষ্য করুন, আমাদের স্ক্যালা ইন্টারপ্রেটারে একটি ফাংশান লিখেছি কিন্তু যেহেতু আমাদেরকে অনেকগুলো লাইনে লিখতে হয়ছে, যে জন্যে প্রত্যেকটি লাইনের পর একটি করে উল্লম্ব বার (|) চলে এসেছে।

ইন্টারপ্রেটার-এ যদি মাল্টিলাইন কোড লিখতে হয়, তাহলে আপনি ইন্টার চেপে যেতে থাকুন। আপনি যদি মনে করে থাকেন আপনি কিছু ভুল লিখে ফেলেছেন তাহলে পর পর দুইবার ইন্টার চাপুন, তাহলে ইন্টারপ্রেটার আপনাকে নতুন কমান্ড লেখার জন্যে কনসোল ফ্রি করে দেবে।

```
scala> val opps =
     |
     |
You typed two blank lines.  Starting a new command.
```

ফাংশান এ ফিরে আসি।

স্ক্যালাতে ফাংশান শুরু হয় def কিওয়ার্ড দিয়ে। তারপর ফাংশান এর নাম। এর পর পেরেন্থেসিস এর মাঝে প্যারামিটার গুলো লিখতে হয়। প্যারামিটার গুলোতে টাইপ নোটেশান আবশ্যক কারণ স্ক্যালা কম্পাইলার মেথড এর টাইপ ইনফার করতে পারে না। এরপার ক্লোজড পেরেন্থেসিস এর পর একটি কোলন(:) দিয়ে পাশে টাইপ নোটেশান লিখতে হয় ।এটি হলো ফাংশানের রিটার্ন টাইপ। এরপর একটি সমান সমান চিহ্নের পর কার্লি ব্রেস এর মাঝে মেথড বডি লিখতে হয়।

ফাংশানটি আবার দেখি-

```
def max(x: Int, y: Int): Int = {
  if (x > y) x
  else y
}
```

খেয়াল করুন, আমরা কোথাও return কিওয়ার্ড ব্যবহার করিনি। জাভাতে আমরা অনেকেই টার্নারি অপারেটর ব্যবহার করি, এখানে ব্যাপরটি তাই। সাধরাণত কোন ফাংশান বডি এর লাস্ট লাইনটি রিটার্ন স্ট্যাটম্যন্ট হিসেবে বিবেচিত হয়। তবে আমরা চাইলে return কিওয়ার্ড করতে পারি, কিন্তু এটি আবশ্যক নয়।

```
def max(x: Int, y: Int): Int = {
  if (x > y) return x
  else  return y
}
```

আরেকটি বিষয় লক্ষ্যনীয়, সেটি হলো আমরা চাইলে রিটার্ন টাইপ নাও লিখতে পারি, সে ক্ষেত্রে স্ক্যালা কম্পাইলার অটোম্যাটিক্যালি ইনফার করে নেবে। উদাহরণ-

```
def max(x: Int, y: Int) = {
  if (x > y) x
  else y
}
```

আমরা আরেকটি উদাহরণ দেখি-

```
scala> def greet() = println("hello world")
greet: ()Unit
```

আমরা এখানে একটি ফাংশান লিখেছি। ইন্টারপ্রেটার এর আউটপুটটি লক্ষ্যনীয়। এখানে greet অবশ্যই ফাংশানের নাম এবং Unit হচ্ছে রিটার্ন টাইপ। এটি মূলত জাভা এর void এর মতো কাজ করে। কোন ফাংশান যদি কোন কিছু রিটার্ন না করে , তাহলে সেটি Unit রিটার্ন করে।

নেক্সট স্টেপস ইন স্ক্যালা

এই চ্যাপ্টার এ আমরা কিছু স্ক্যালা কন্ট্রোল-ফ্লো নিয়ে আলোচনা করবো -

খুব ইন্টারেস্টিং কিছু নাও মনে হতে, শুধু মাত্র কিছু ভ্যারিয়েবল ডিক্ল্যারেশান এবং এক্সপ্রেশান এর ডেমনস্ট্রেশান। তাহলে আমরা আমাদের ইন্টারপ্রেটার-এ ফিরে যাই-

শুরুতে আমরা একটি সিম্পল এক্সপ্রেশান লিখি - একটি সংখ্যা এবং আরেকটি সংখ্যা যোগ করি । ইতপূর্বে আমরা যদিও করেছি-

```
scala> 1 + 1
res0: Int = 2
```

একটু সফিস্তিকেটেড কোড লিখতে হলে আমাদের ভ্যারিয়েবল এর দরকার হয়।

```
scala> var x: Int = 1 + 1
x: Int = 2
```

আপনি যদি জাভা কিংবা সি প্রোগ্রামিং এ অভ্যস্ত হয়ে থাকেন তাহলে একটু উদ্ভট মনে হতে পারে, কারণ আমরা আগে টাইপ ইনফরমেশান লিখি তারপর ভ্যারিয়বল এর নাম দেই। কিন্তু স্ক্যালা এর ক্ষেত্রে একটু উল্টোভাবে লিখি। এক্ষেত্রে আমরা var x লিখি যা নির্দেশ করে যে আমরা একটি ভ্যারিয়বল লিখতে যাচ্ছি। তারপর আমরা সেই ভ্যারিয়বল কে ascribe করি, অর্থাৎ টাইপ ইনফরমেশান ডেসক্রিপশান লিখি এবং তারপর সমান সমান চিহ্ন দিয়ে আমাদের এক্সপ্রেশান লিখি।

এখানে var বলতে বুঝায় x একটি মিউটেবল ভ্যারিয়েবল অর্থাৎ এর মান আমরা পরিবর্তন করতে পারি।

```
scala> x = 5
x: Int = 5
```

এবার অন্য ভ্যারিয়েবল দিয়ে চেষ্টা করি-

```
scala> var s : String = "Hello "
s: String = "Hello "
```

আমরা স্ট্রিং Concatenate করতে পারি-

```
scala> s = s + "scala!"
s: String = Hello scala!
```

আমরা এবার অন্যান্য ল্যাংগুয়েজর এর মতো রেগুলার কন্ট্রোল ফ্লো দেখি -

```
scala> if (true) x = x + 12 else x = x * 13
```

এক্ষেত্রে কনসোল কোন কিছু প্রিন্ট করে নি। এটি একটি রেগুলার কন্ট্রোল - ফ্লো আমরা যদি এখন x এর মান দেখতে চাই, তাহলে -

```
scala> x
res3: Int = 17
```

আমরা একটু ইন্টারেস্টিং জিনিস দেখি। স্ক্যালাতে if – else শুধুমাত্র একটা স্টেটমেন্ট হিসেবে ব্যবহার না করে একে এক্সপ্রেশান হিসেবে ব্যবহার করা যায়।

```
scala> x = if (true) x + 12 else x * 13
x: Int = 29
```

এর মানে, x হতে পারে x + 12 অথবা x * 13 এটি নির্ভর করে কন্ডিশন ভ্যালু এর উপর।

ইন্টারেস্টিং। আমরা চাইলে এখন এই এক্সপ্রেশান ব্যবহার করে আরও কমপ্লেক্স এবং সফিস্তিকেটেড এক্সপ্রেশান তৈরি করতে পারি।

```
scala> var x = 5
x: Int = 5

scala> var y = if (false) {
     |    x + 1
     | } else {
     |    if (true) {
     |       x - 1
     |    } else {
     |      x -2
     |    }
     | }
```

ইন্টারেষ্টিং লজিকাল কুনানড্রাম !

এখন y এর মান কত হতে পারে ?

যেহেতু , শুরুর if ব্লক-এ false তাই এটি else ব্লকে যাবে, এবং এখানের if ব্লক true তাই রেজাল্ট হবে x – 1

অর্থাৎ -

```
y: Int = 4
```

আমরা জানি আরও এক ভাবে ভ্যারিয়েবল ডিক্ল্যায়ার করা যায় – সেটি হলো val

```
scala> val z = 1 + 1
z: Int = 2
```

এখানে z হল- ইমিউটেবল ভ্যারিয়েবল ,অর্থাৎ আমরা চাইলে এতে ভ্যালু রি-এসাইন করতে পারবো । এবং আমরা যদি করতেও চাই, সাথে সাথে কম্পাইলার ইরর দেখাবে -

```
  scala> z = z + 1
  <console>:8: error: reassignment to val
         z = z + 1
           ^
```

কিন্তু আমি এখানে একটা জিনিস মিস করে গেছি । একটু খেয়াল করলেই দেখা যাবে যে var y = … এই এক্সপ্রেশান এ কোন টাইপ ডেসক্রিপশান লিখিনি। কিন্তু তার পরেও এটি ঠিকঠাক মতো কাজ করে গেছে। আমরা ভাল করেই জানি যে স্ক্যালা স্ট্যাটিক ল্যাংগুয়েজ এবং কম্পাইল ল্যাংগুয়েজ। সুতরাং কম্পাইলার কম্পাইল টাইম এ সব স্ট্যাটিক্যালি টাইপ চেক করার কথা। কিন্তু এখানে আমরা কোন টাইপ দেইনি বলে ইরর দেখানোর কথা ছিল। কিন্তু স্ক্যালা কম্পাইলার তা করে নি। স্ক্যালা অন্যান্য স্ট্যাটিক ল্যাংগুয়েজ যেমন- জাভা, সি এর মতোই। কিন্তু স্ক্যালাতে একটি মজার জিনিস আছে যাকে আমরা বলি টাইপ ইনফারেন্স(Type Inference)। স্ক্যালা এক্সপ্রেশান এর টাইপ থেকে এখানে y ভ্যারিয়েবল এর টাইপ ইনফার করতে পারে।

নিচের উদাহরণটি দেখি -

```
    scala> var x = 1
```

এখানে এক্সপ্রেশান হচ্ছে ভ্যালু 1 এবং এটি ইন্টিজার। সুতরাং এটি থেকেই বুঝা যাচ্ছে যে x টাইপ হবে Int.

এভাবে অন্যান্য টাইপ এর ক্ষত্রেও এটি টাইপ ইনফার করতে পারে।

এখানে আরো একটি ইন্টারেস্টিং জিনিস খেয়াল করি - আমাদের এক্সপ্রেশান যদি এমন হয়-

```
scala> :t if (false) "hello" else 1
Any
```

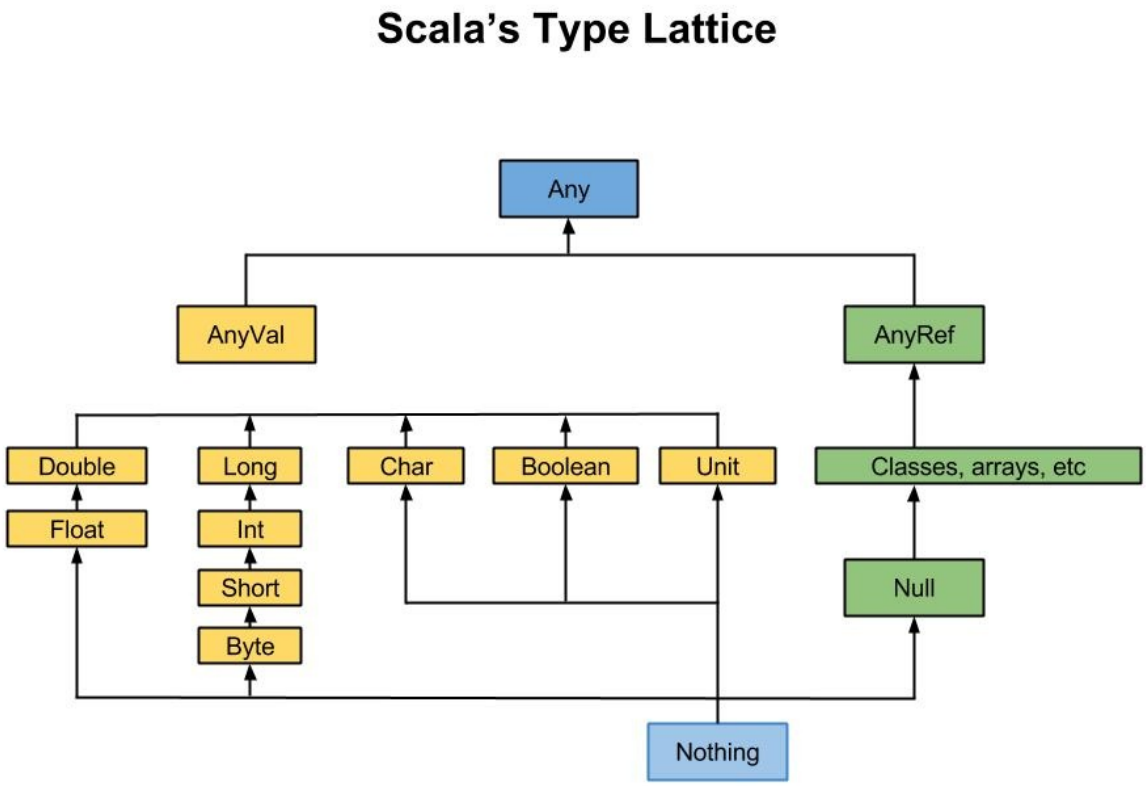
অথবা -

```
scala> :t if (true) "hello" else 1
Any
```

Note: এখানে :t এক স্পেশাল অপারেটর যা দিয়ে আমরা টাইপ ডেসক্রিপশান বের করতে পারি।

এখানে দেখা যাচ্ছে যে দুটি ক্ষেত্রেই টাইপ হচ্ছে Any

এর মানে কি? এর উত্তর দেখতে হলে আমাদের দেখতে হবে -**scala type lattice**



এটি হলো স্ক্যালা এর টাইপ হায়ারার্কি। এই হায়ারার্কি এর একদম উপরে আছে Any। এটি দুই প্রকর হতে পারে। ভ্যালু টাইপ এবং রেফারেন্স টাইপ। AnyVal হচ্ছে সকল প্রিমিটিভ টাইপ এর সাব ক্লাস । অর্থাৎ এগুলো হচ্ছে বিল্ট-ইন টাইপ যেগুলো JVM এ থাকে - যেমন Floating point, integer ইত্যাদি। তবে এখানে ডায়াগ্রাম এ Byte আসলে Short কে extends করে না এভাবে Short আসলে Int কে extends করে না। এটি শুধুমাত্র ডায়াগ্রাম তৈরির সুবিধার্থে করা হয়েছে। আর ডান পাশের গুলো হচ্ছে রেফারেন্স টাইপ – ক্লাস , অ্যারে ইত্যাদি, যেমন String, List . এখানে একটা জিনিস বলে নেই, সব গুলো টাইপ এর একটি কমন সাব ক্লাস আছে, সেটি হলো - Null. অর্থাৎ যেকোন টাইপ আসলে Null হতে পারে।

নিচের একটি চার্ট দেওয়া হলো-

```
Data Type|    Description
-------------| ----------------------
Byte|        8 bit signed value. Range from -128 to 127
Short|        16 bit signed value. Range -32768 to 32767
Int|        32 bit signed value. Range -2147483648 to 2147483647
Long|        64 bit signed value. -9223372036854775808 to 9223372036854775807
Float|        32 bit IEEE 754 single-precision float
Double|        64 bit IEEE 754 double-precision float
Char    |    16 bit unsigned Unicode character. Range from U+0000 to U+FFFF
String    |    A sequence of Chars
Boolean|     Either the literal true or the literal false
Unit|        Corresponds to no value
Null|        null or empty reference
Nothing |    The subtype of every other type; includes no values
Any    |    The supertype of any type; any object is of type Any
AnyRef |    The supertype of any reference type
```

এবার আমরা আবার ইন্টারপ্রেটারে ফিরে যাই -

```
    scala> :t if (false) "hello" else 1
    Any
```

এবং

```
    scala> :t if (true) "hello" else 1
    Any
```

এর দুটির টাইপ এখানে Any। কেন এর উত্তর আমরা ল্যাটিস থেকেই দেখতে পারছি যে, এদের কমন ancestor হচ্ছে - Any

আমরা আরও কয়েকটি উদাহরণ দেখি -

```
    scala> :t if(true) 1 else 1.0
    Double
```

লক্ষ করি, এটির টাইপ হচ্ছে Double কারণ Int কে কনভার্ট করে Double বানানো যায়।

এবার আমরা অন্য একটি বিষয় লক্ষ্য করি -

```
scala> val s:String = if(true) "hello" else 1
<console>:7: error: type mismatch;
 found   : Int(1)
 required: String
       val s:String = if(true) "hello" else 1
```

আমরা দেখতে পাচ্ছি যে , এখানে s এর টাইপ হচ্ছে String কিন্তু এক্সপ্রেশান থেকে তা String অথবা Int যে কোনটি হতে পারে, এবং সিটি নির্ধিরণ হবে রানটাইম-এ কিন্তু যেহেতু স্ক্যালা একটি কম্পাইল্ড ল্যাংগুয়েজ, তাই এটি type mismatch ইরর দিচ্ছে।

সুতরাং যে বিষয় গুলো এখানে লক্ষ্য করতে হবে তা হচ্ছে- এক্সপ্রেশান যদিও ভ্যালু তৈরি করে এবং রানটাইম-এ কম্পিউট হয়, তারপরেও এর একটি টাইপ থাকে যা নির্ধারণ হয় কম্পাইল-টাইম এ। যদিও স্ক্যালা টাইপ ইনফারেন্স চমৎকার , তারপরেও আমাদের মাথায় টাইপ ব্যপারটি রাখতে হবে। কারণ এটি মোটেও কোন ডাইনামিক ল্যাংগুয়েজ নয়।

আমরা যেহেতু ভ্যারিয়েবল ব্যবহার করতে পারি, সুতরাং ফাংশন নিয়ে কথা বলি।

নিচের ফাংশনটি দেখি -

```
scala> def add(a: Int, b: Int): Int =  a + b
add: (a: Int, b: Int) Int

scala> add(3,4)
res3: Int = 7
```

দেখা যাচ্ছে যে এটি দুটি Int টাইপ ডাটা নিয়ে তা যোগ করে রিটার্ন করে।

কিন্তু এই ফাংশনটি যদি এভাবে লিখি -

```
scala> def add(a , b) =  a + b

<console>:1: error: ':' expected but ',' found.
       def add(a , b) =  a + b
                 ^
```

দেখা যাচ্ছে যে, এটি কাজ করছে না । যদিও আমরা a + b থেকে বুঝে নিতে পারি এটি দুটি Int হতে পারে। কিন্তু ব্যপারটি হচ্ছে a + b দুটি Double অথবা Float ও হতে পারে।

সুতরাং দেখা যাচ্ছে স্ক্যালা তে প্যারামিটার এ টাইপ ইনফার করতে পারছে না। সত্যি বলতে কি, স্ক্যালা পার্সার এটি ফোর্স করে। অর্থাৎ স্ক্যালতে প্যারামিটার এ টাইপ এসক্রিপশান থাকতে হবে।

তবে আমরা চাইলে রিটার্ন টাইপ নাও দিতে পারি। এক্ষেত্রে স্ক্যালা এটিকে ইনফার করে নিতে পারে। উদাহরণ-

```
scala> def add(a: Int, b: Int) =  a + b
add: (a: Int, b: Int)Int

scala> add( 3, 4)
res4: Int = 7
```

কারন এক্ষত্রে আমরা যেহেতু a এবং b এর টাইপ জানি, সুতরাং এটির এক্সপ্রেশান থেকে টাইপ ইনফার করা যাচ্ছে।

তবে কিছু সিচুয়েশান আছে যেখানে স্ক্যালা রিটার্ন টাইপ ইনফার করতে পারে না । বিশেষত রিকার্সিভ মেথড গুলোর ক্ষেত্রে।

উদাহরণসরূপ ফিবোনাচি নাম্বার এর জন্যে একটি ফাংশান লিখি -

```
scala> def fib(n: Int) = if (n==1 || n ==2) 1 else fib(n -1) + fib(n-2)
<console>:7: error: recursive method fib needs result type
       def fib(n: Int) = if (n==1 || n ==2) 1 else fib(n -1) + fib(n-2)
                                                    ^
```

সুতরাং দেখা যাচ্ছে এক্ষেত্রে এটি রিটার্ন টাইপ ইনফার করতে পারে নি। কিন্তু যদি আমরা রিটার্ন টাইপ দিয়ে দিই, তাহলে এটি খুব ভালভাবে কাজ করে –

```
scala> def fib(n: Int): Int = if (n==1 || n ==2) 1 else fib(n -1) + fib(n-2)
fib: (n: Int)Int

scala> fib(2)
res5: Int = 1

scala> fib(3)
res6: Int = 2

scala> fib(4)
res7: Int = 3

scala> fib(5)
res8: Int = 5

scala> fib(6)
res9: Int = 8
```

সুতরাং মনে রাখতে হবে যে, রিকার্সিভ ফাংশান এর ক্ষেত্রে অবশ্যই রিটার্ন টাইপ দিতে হবে।