Programming Language Paradigms

Programming Language Theory

Topics

- Midterm Exam Review
- **PL Paradigm Overview**
- Scripting Language

PL Paradigm

- Principles and Strategies which a programming language follows.
    - e.g.) Procedural, Imperative, OOP, Functional, Logic, etc.

- One programming language may have adopted multiple paradigms.

    - Java - Imperative and Object-Oriented.

Language Paradigms

- So many ways to group or categorize programming languages.
- **Imperative**
    - procedural (Fortran, Pascal, Basic, C)
    - object-oriented (Smalltalk, C++, Java)
    - scripting languages (Perl, Python, JavaScript)
- **Declarative**
    - functional programming (Scheme, ML, Haskel)
    - logic programming (Prolog, Datalog)

Imperative Languages

- *imperative*: *giving an authoritative* ***command (or statements)****.*

- Using ***assignments*** to change a program's state.

- Focus on ***how*** a program operates (algorithms).

- In low-level, machine code (instructions) is to modify the program state (i.e. register/memory).

- In high-level, program states are described by variables, and statements are the machine instructions.

Declarative Languages

- Declarative languages are often defined as ***non-imperative*** languages.

- Instead of how to do, it describes ***what to do***, or ***what a program should be***.

- Programs describe ***desired results***, rather than provide commands which should be performed by a computer.

- *Functional* and *Logic* languages.

- However, ***there is no clear distinction*** between Imperative and Declarative languages.

Example: Euclidean Algorithm

- **Imperative: C++**
- Start from main(), the program is a series of commands.
- We can follow the program step by step.

```cpp
int gcd(int a, int b) {
    if(b == 0)
        return a;
    else
        return gcd(b, a%b);
}

int main() {
    cout << gcd(10, 5);
    return 0;
}
```

Example: Euclidean Algorithm

- **Imperative, OOP: Java**
- We can also follow the program step by step with main() part.
  - Imperative.
- The program defines a class GCD, and hides its information inside, provides operations for the class.
  - OOP.

```java
public class GCD {
    private int a, b;
    public GCD(int a, int b) {
        this.a = a;
        this.b = b;
    }
    public int compute() {
        return a >= b ? gcd(a, b) : gcd(b, a);
    }
    private int gcd(int a, int b) {
        if(b == 0)
            return a;
        else
            return gcd(b, a%b);
    }
    Run | Debug
    public static void main(String[] args) {
        System.out.println(new GCD(12, 4).compute());
    }
}
```

Example: Euclidean Algorithm

- **Declarative, Functional: Scheme**

- It simply defines what is a function gcd.
  - Declarative.
- Still, it has a command to print something to screen.
  - Imperative.

```
(define (gcd a b)
    (if (equal? b 0)
        a
        (gcd b (modulo a b))))
(display (gcd 12 4))
```

Example: Euclidean Algorithm

- **Declarative, Functional: Scala**

- It defines what is a function gcd.
  - Declarative.

- It also contains Java-like features such as object GCD, main() method.

```
object GCD {
  def main(args: Array[String]): Unit = {
    println(gcd(12, 4))
  }
  def gcd(a: Int, b: Int): Int =
    if (a%b==0)
      b
    else {
      var r: Int = a%b
      gcd(b, a%b)
    }
}
```

Example: Euclidean Algorithm

- **Declarative, Logic: Prolog**
- It describes ***Facts*** and ***Rules*** about gcd.
  - Declarative.
- You can ask ***questions (or queries)*** based on the facts and rules.

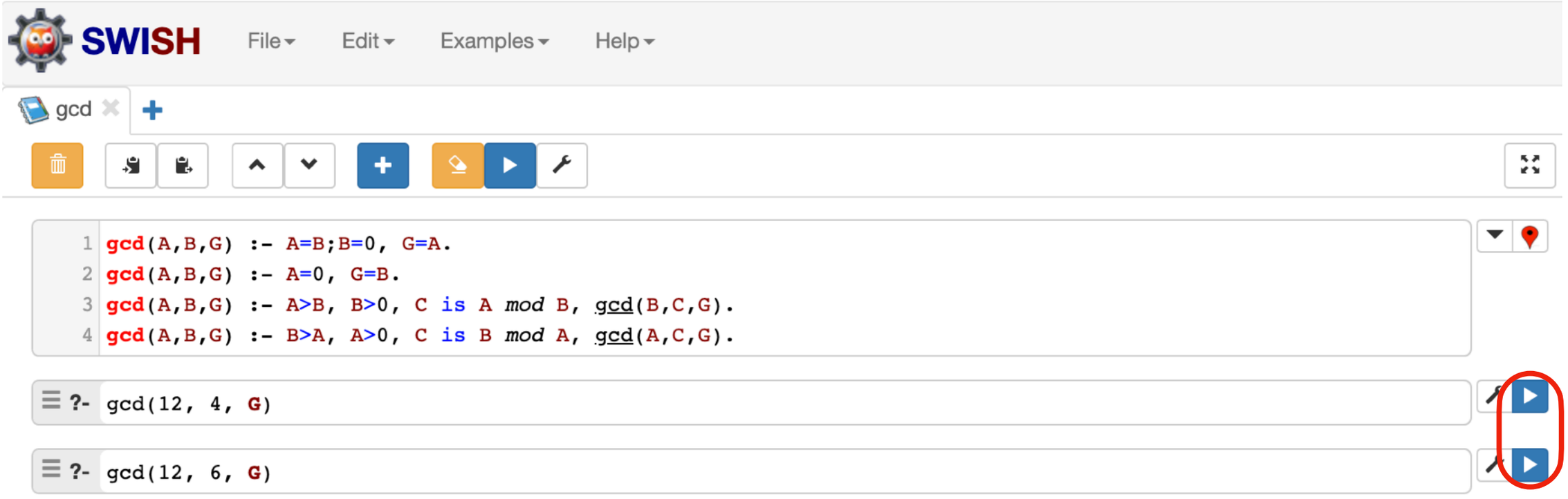
```
gcd(A,B,G) :- A = B;B = 0, G = A.
gcd(A,B,G) :- A = 0, G = B.
gcd(A,B,G) :- A > B, C is A mod B, gcd(C,B,G).
gcd(A,B,G) :- A < B, C is B mod A, gcd(C,A,G).
```

```
?- gcd(12, 4, G).
G = 4 .

?- gcd(12, 6, G).
G = 6 .
```

Example: Euclidean Algorithm

https://swish.swi-prolog.org/p/gcd.swinb



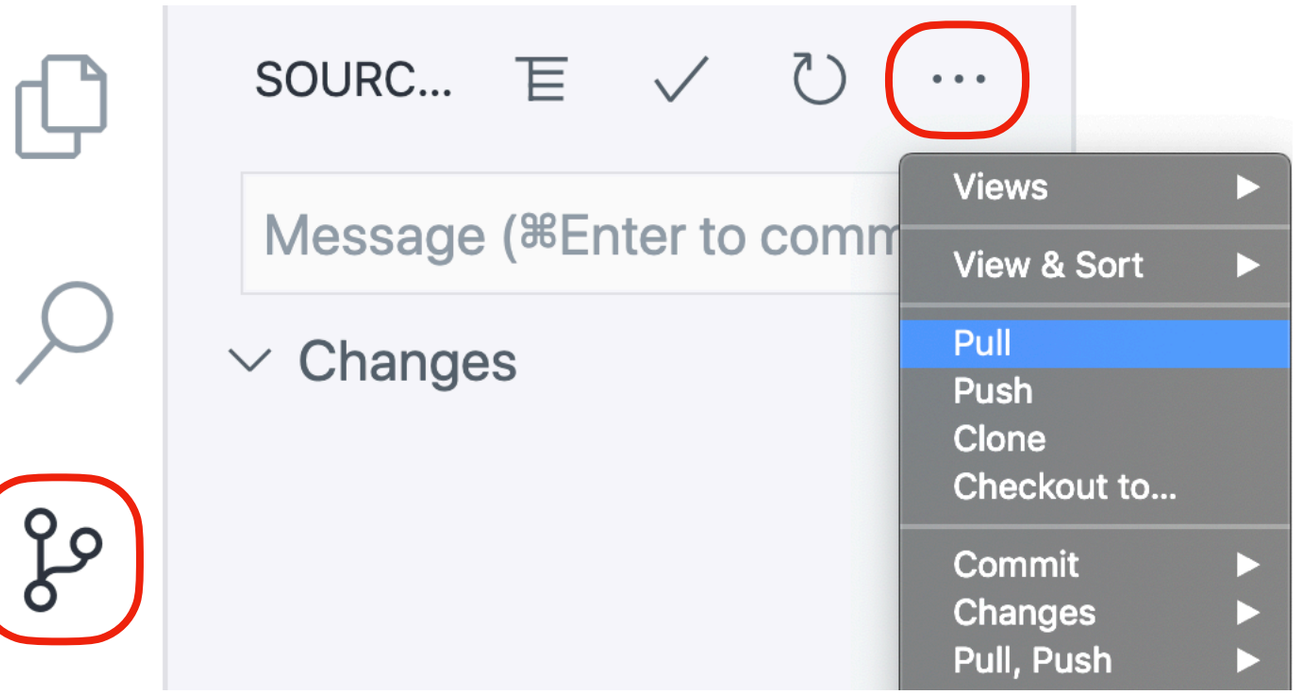
Examples from GitHub

- All the examples are available as practices3 in Course GitHub.

- You can download files by command 'Git Pull' in VSCode.

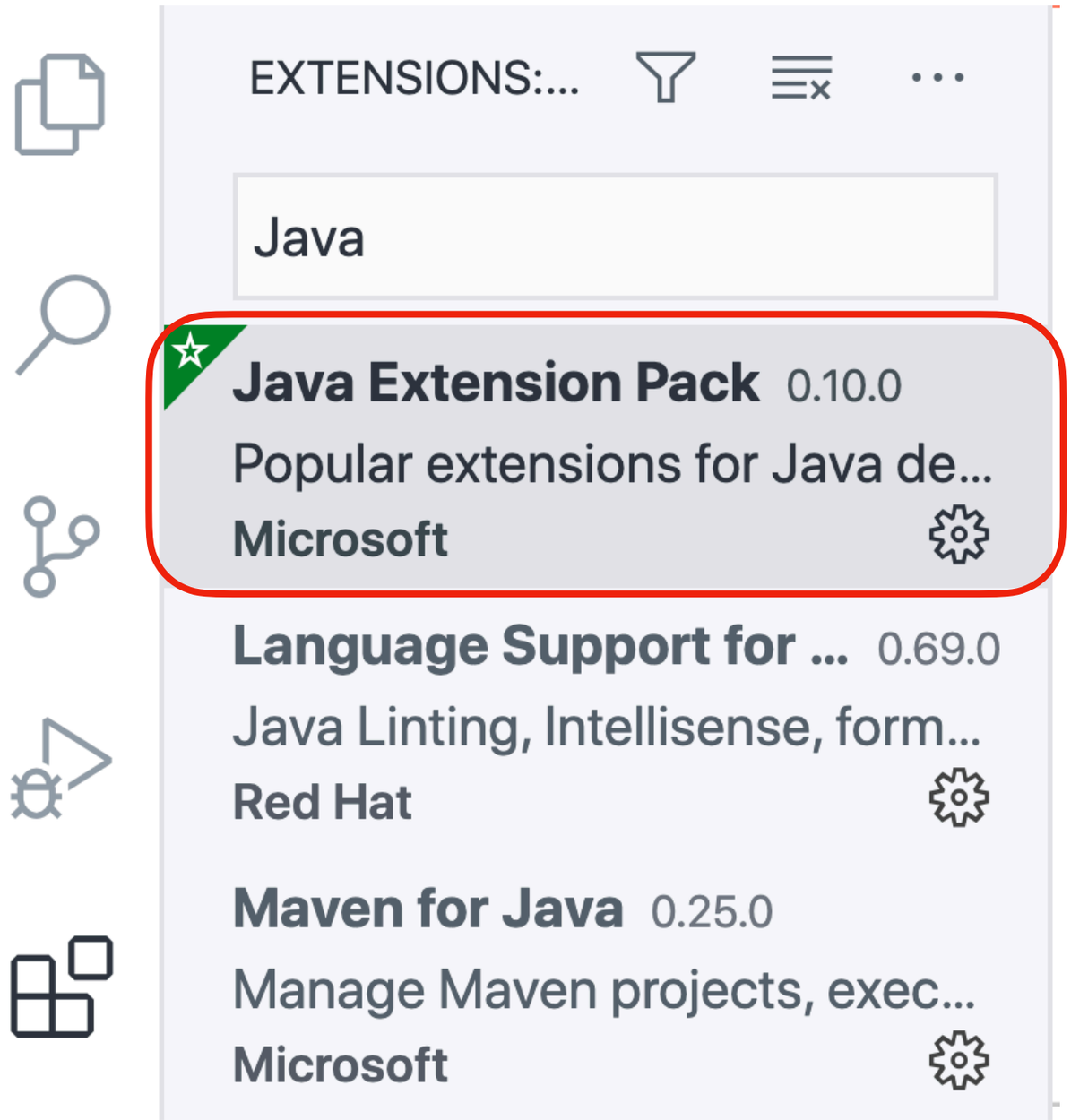
- However, executing examples is another story.

  - You need a compiler or an interpreter for each language.



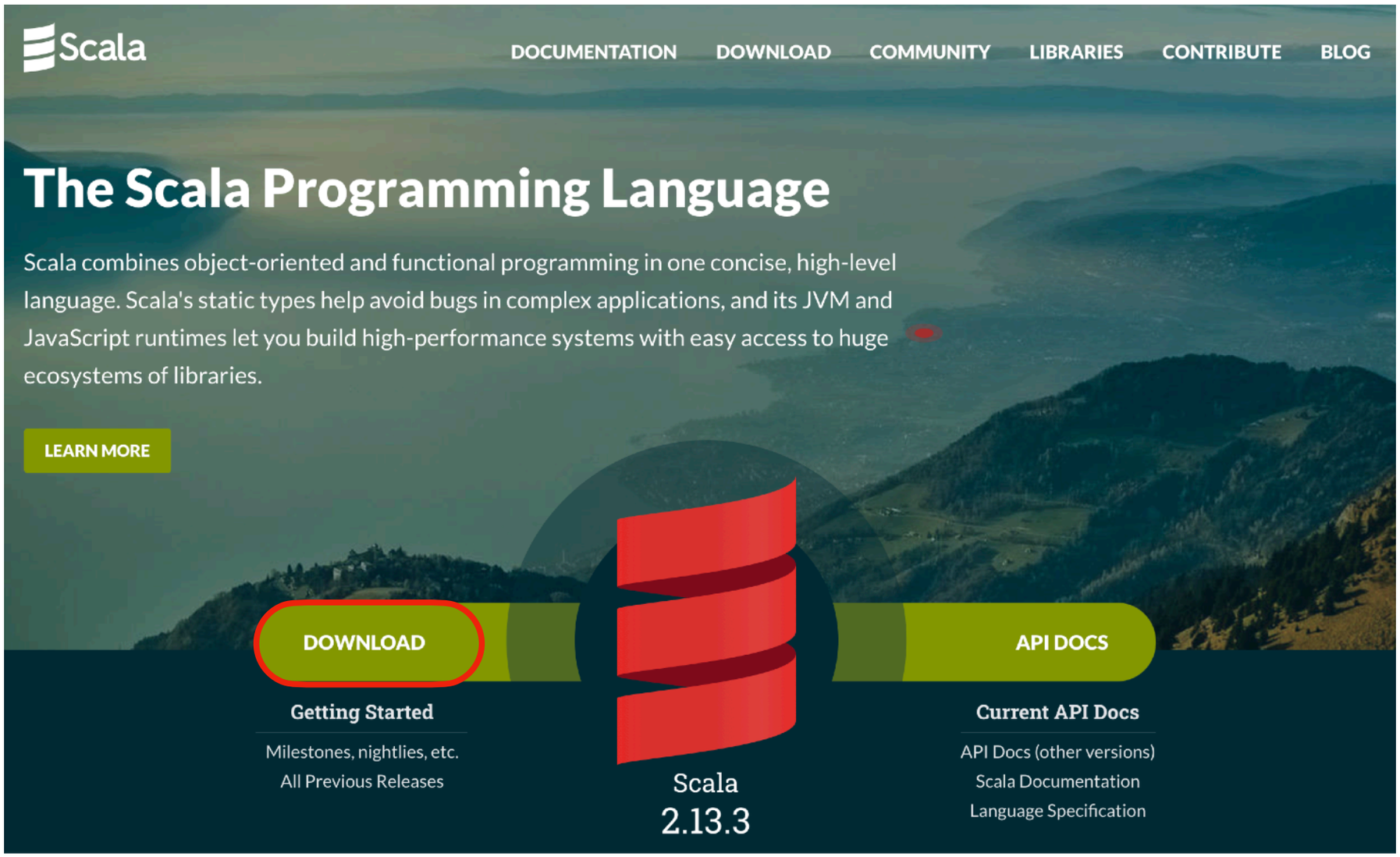
C++ and Java

- We already installed extension for C, C++ at the first practice.
- For Java, you must install Java Development Kit first.
  - https://openjdk.java.net/install/
  - https://www.oracle.com/java/technologies/javase-downloads.html
- You can install Java extension in VSCode.
- Or postpone setup until setup Scala.



Scala

https://www.scala-lang.org/



Scala

The most popular way to get Scala is either using Scala through sbt, the Scala build tool, or to use Scala through an IDE.

1

First, make sure you have the Java 8 JDK (or Java 11 JDK) installed.

To check, open the terminal and type:

`java -version` *(Make sure you have version 1.8 or 11.)*

(If you don't have it installed, download Java from Oracle Java 8, Oracle Java 11, or AdoptOpenJDK 8/11. Refer JDK Compatibility for Scala/Java compatiblity detail.

Scala

2

Then, install Scala:

...either by installing an IDE such as IntelliJ, or sbt, Scala's build tool.

DOWNLOAD INTELLIJ or **DOWNLOAD SBT**

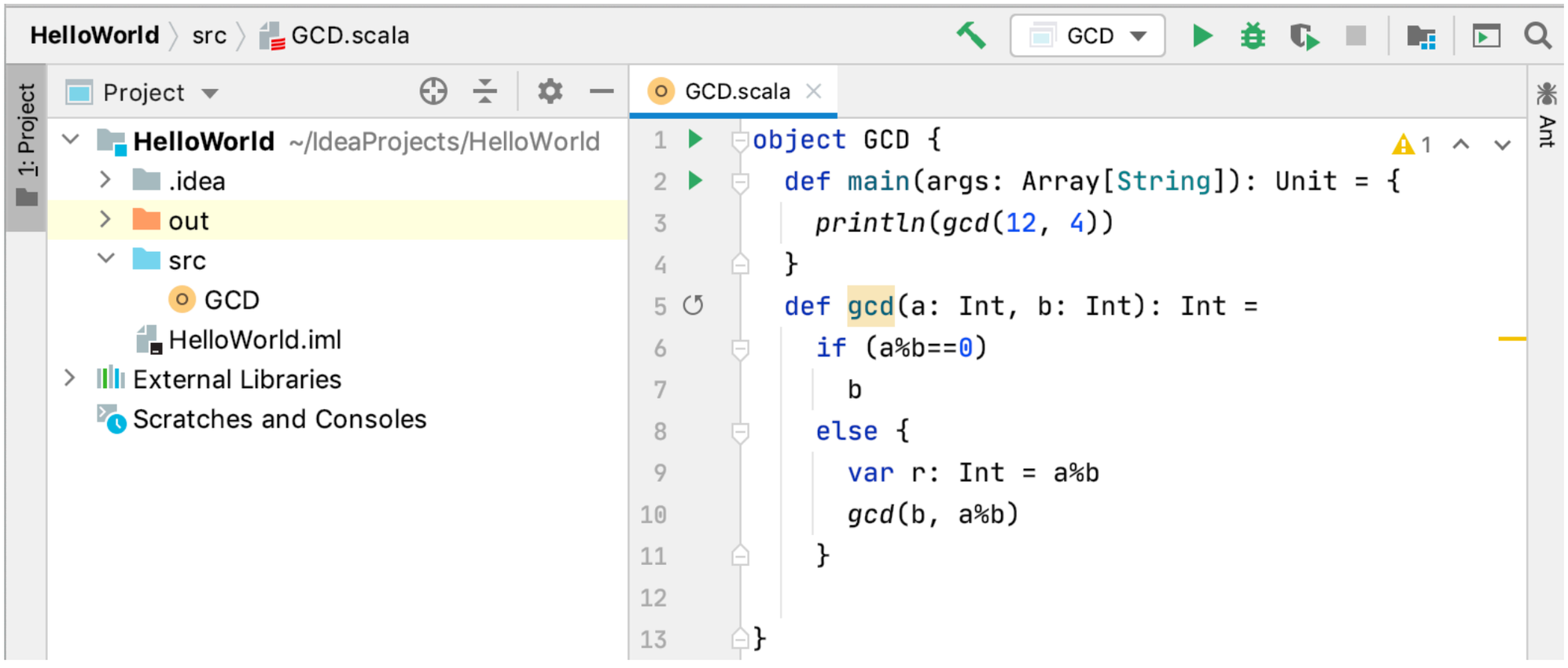
- Getting Started with Scala in IntelliJ
- Building a Scala Project with IntelliJ and sbt
- Testing Scala in IntelliJ with ScalaTest

Best if you prefer a full-featured IDE (recommended for beginners)

- Getting Started with Scala and sbt on the Command Line
- Testing Scala with sbt and ScalaTest on the Command Line

Best if you are familiar with the command line

Scala with IntelliJ



Scheme

https://www.call-cc.org/

CHICKENscheme

a practical and portable scheme system

Wiki | Download | Manual | Eggs | API | Tests | Bugs

CHICKEN is a compiler for the Scheme programming language. It produces portable and efficient C and supports the R5RS and R7RS (work in progress) standards, and many extensions. It runs on Linux, OS X, Windows, many Unix flavours, and aims to be...

Free

Distributed free for use and modification under the BSD License (freedom to user and developer).

Simple

Lightweight on dependencies (a C toolchain and GNU Make) and easy to install.

Portable

Runs on many platforms, including x86, x86-64, ARM, MIPS, SPARC64 and PowerPC.

Extensible

Many libraries and extensions are available at Eggs Unlimited and new ones can be easily created.

Well documented

Wiki, Manual and API reference provide updated and detailed documentation about CHICKEN use.

Actively supported

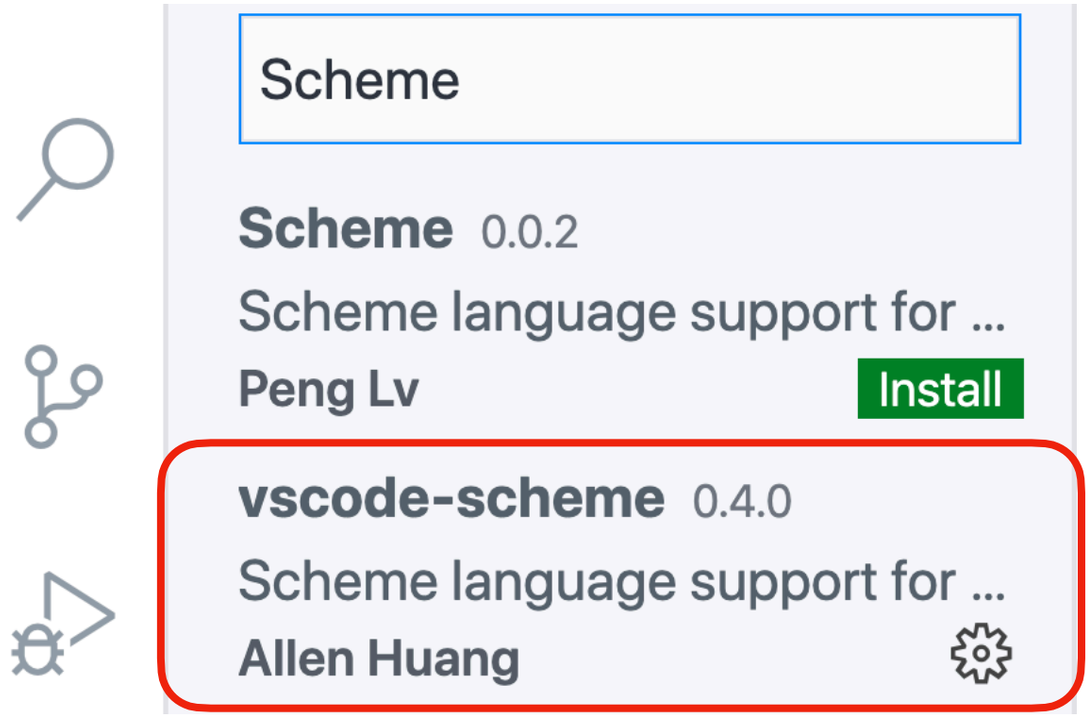
Mailing lists and an IRC channel (**#chicken** on Freenode) for useful and kind support.

Scheme Extension

- Install ***vscode-scheme*** Extension.

- Setup running tasks in ***tasks.json***.

- If you did the setup in the first practice, you can execute a scheme file with the shortcut you've used (e.g. ***Ctrl + Alt + R***), then select "run scheme".




```
{
    "label": "run scheme",
    "type": "shell",
    "command": "csi",
    "args": ["-s", "${file}"],
    "group": {
        "kind": "test",
        "isDefault": true
    }
}
```

Code Runner Extension

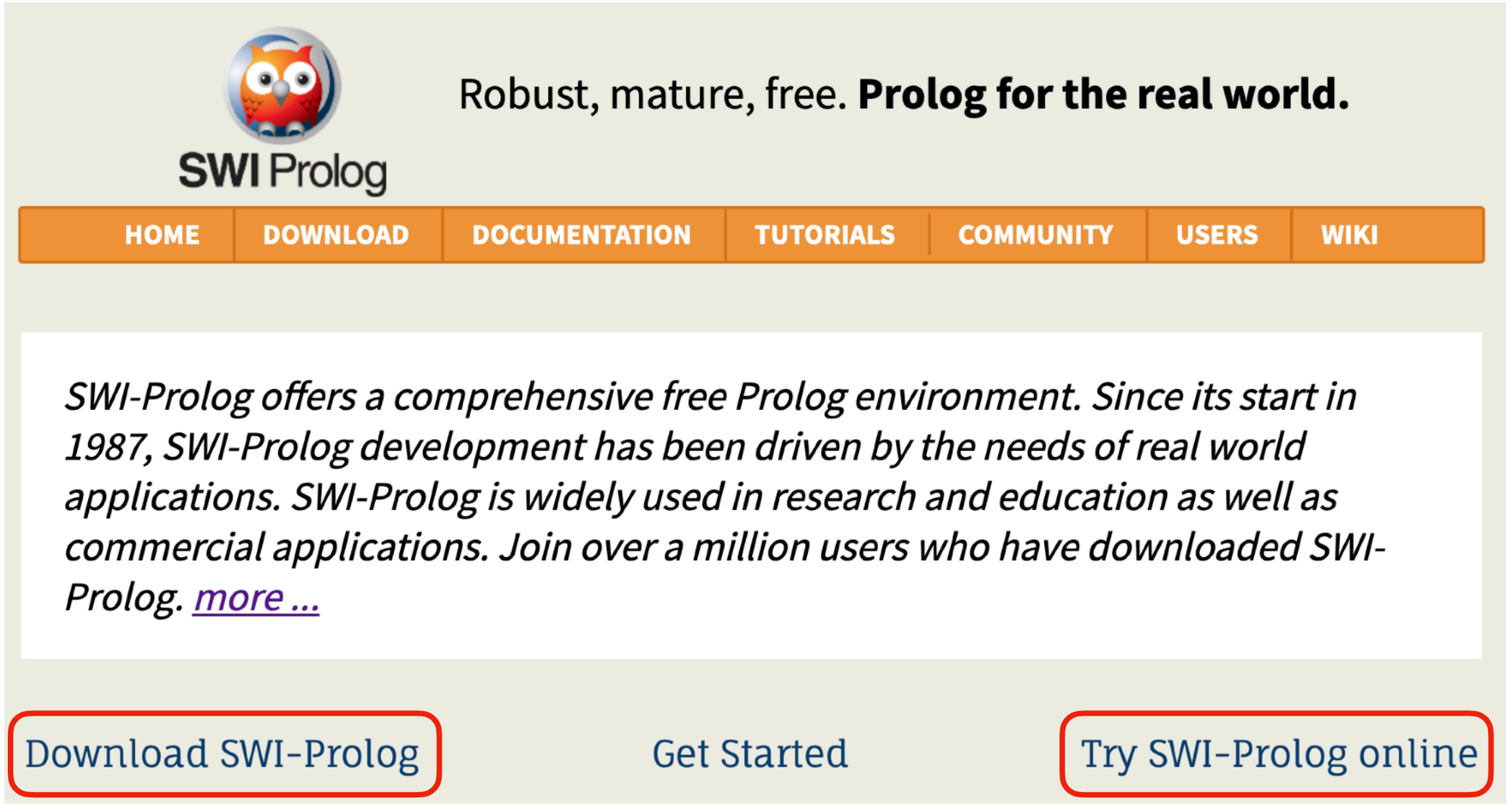
- You can install Code Runner Extension.

- Then setup the executor path in its setting.

- In code editor, you can right click > Run Code.
  - Or Ctrl + Alt + N.

```
{
    "code-runner.executorMap": {
        "javascript": "node",
        "php": "C:\\php\\php.exe",
        "python": "python",
        "perl": "perl",
        "ruby": "C:\\Ruby23-x64\\bin\\ruby
        "go": "go run",
        "html": "\"C:\\Program Files (x86)
        "java": "cd $dir && javac $fileNam
        "c": "cd $dir && gcc $fileName -o
    }
}
```

SWI-Prolog


SWI Prolog
Robust, mature, free. Prolog for the real world.
HOME
DOWNLOAD
DOCUMENTATION
TUTORIALS
COMMUNITY
USERS
WIKI
SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. more ...
Download SWI-Prolog
Get Started
Try SWI-Prolog online

Download SWI-Prolog stable versions

HOME | DOWNLOAD | DOCUMENTATION | TUTORIALS | COMMUNITY | USERS | WIKI

Linux versions are often available as a package for your distribution. We collect information about available packages and issues for building on specific distros here. We provide a PPA for Ubuntu and snap images

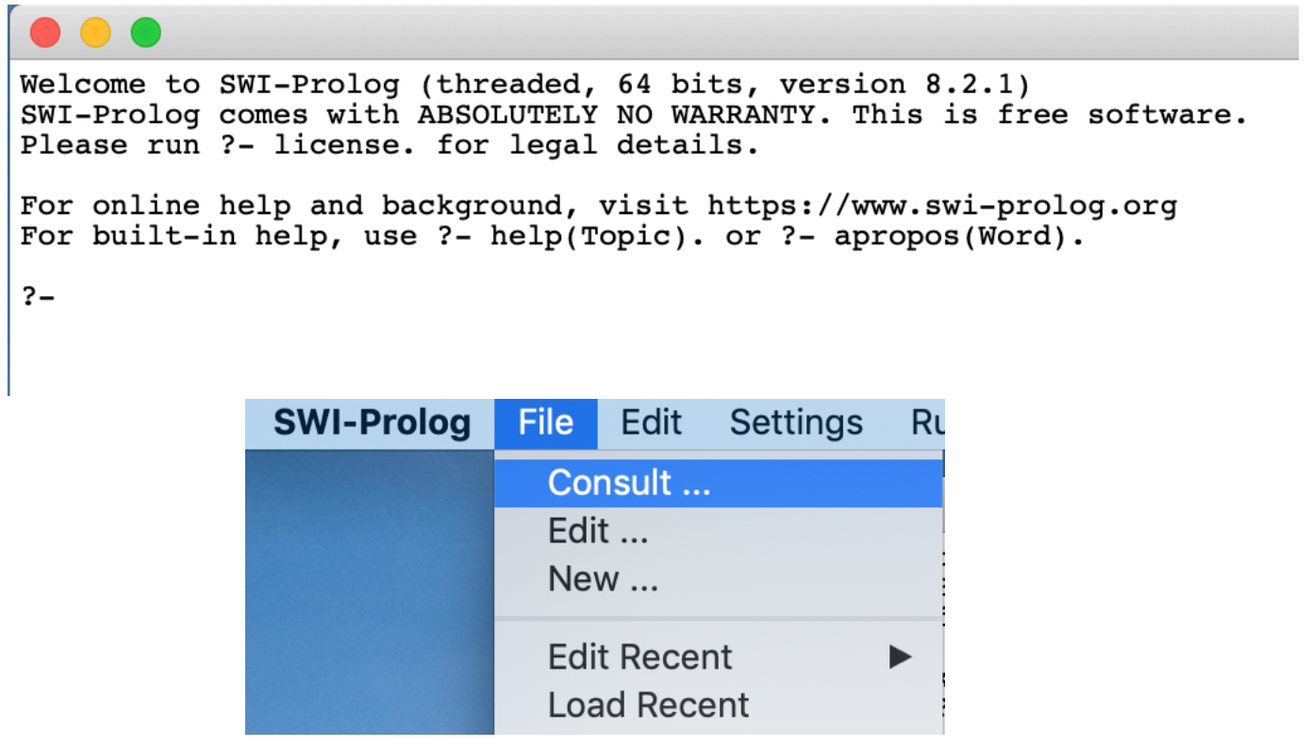
Please check the windows release notes (also in the SWI-Prolog startup menu of your installed version) for details.

Examine the ChangeLog.

Binaries		
	12,810,386 bytes	SWI-Prolog **8.2.1-1** for Microsoft Windows (64 bit) Self-installing executable for Microsoft's Windows 64-bit editions. Requires at least Windows 7. See the reference manual for deciding on whether to use the 32- or 64-bits version. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. **SHA256**: `45fc082c3f39188657e21c0bd51173a0be1bdb7528f584913ac7f18b49fd0cb6`
	12,460,084 bytes	SWI-Prolog **8.2.1-1** for Microsoft Windows (32 bit) Self-installing executable for MS-Windows. Requires at least Windows 7. Installs **swipl-win.exe** and **swipl.exe**. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. **SHA256**: `38537e2cd70630d65a9868708fd305c5600d654fb736d1dbc16090fe9eb2894d`
	27,697,698 bytes	SWI-Prolog **8.2.1-1** for MacOSX 10.12 (Sierra) and later on **intel** Installer with binaries created using Macports. Installs `/opt/local/bin/swipl`. Needs xquartz (X11) and the Developer Tools (Xcode) installed for running the development tools **SHA256**: `1fd495fea2e523b098c7221c092fb6403cbeed7f9c99df3737cd336bb39d6b84`
Sources		
	10,969,688 bytes	SWI-Prolog source for **8.2.1** Sources in `.tar.gz` format, including packages and generated documentation files. See build instructions. **SHA256**: `331bc5093d72af0c9f18fc9ed83b88ef9ddec0c8d379e6c49fa43739c8bda2fb`

SWI-Prolog

- After installing it, you can launch SWI-Prolog app.



- Then you can load the example file "gcd.pl" using File > Consult ... command.

- Once the file is loaded, you can ask your queries in the console.

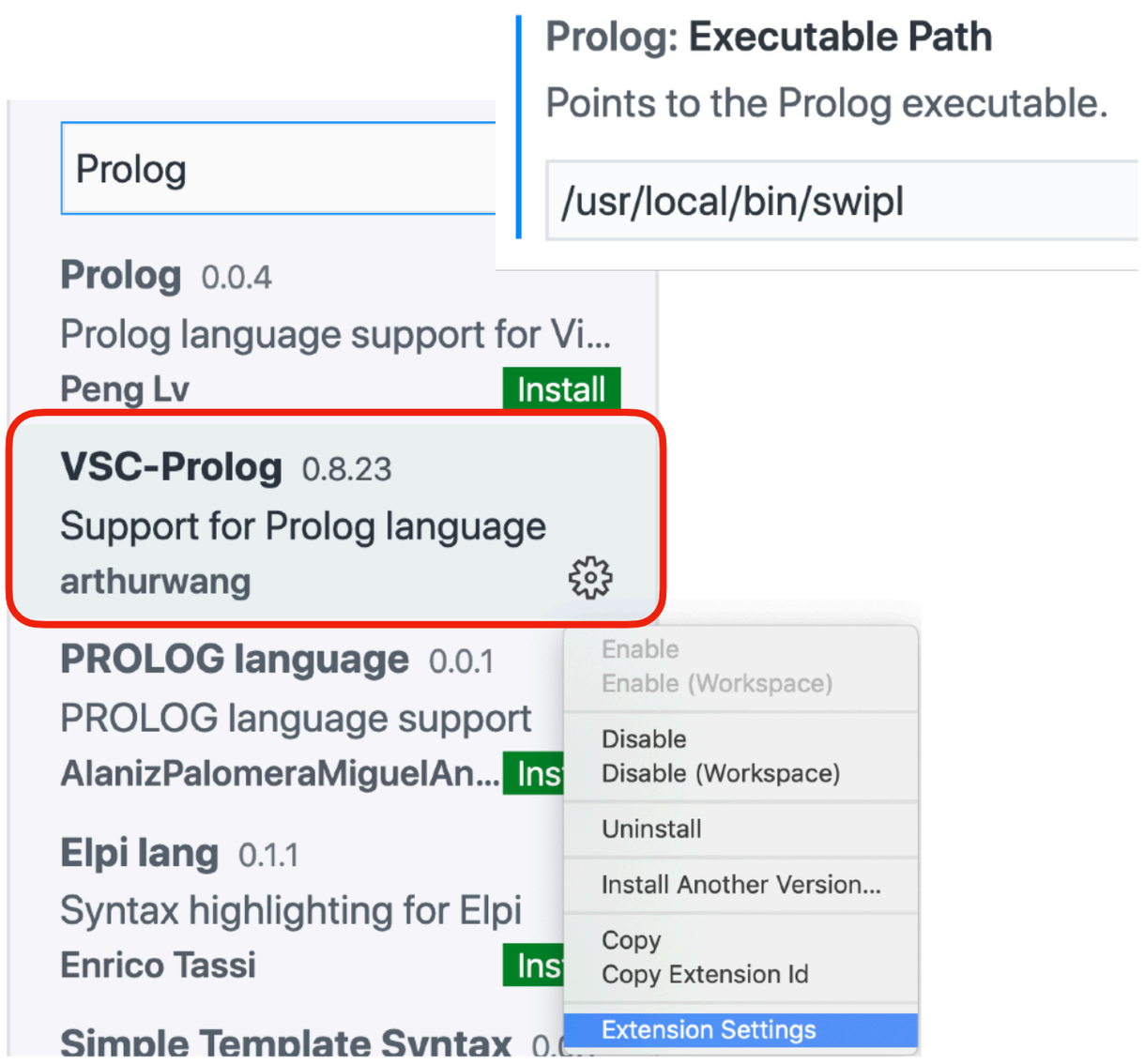
```
?- gcd(12, 4, G).
G = 4 .

?- gcd(12, 6, G).
G = 6 .

?-
```

VSCode Extension

- In Extension tab, search for Prolog.
- Install VSC-Prolog extension.
- Then click the Gear icon, then click Extension Settings.
- Put the correct executable path you've installed.



Execution

gcd.pl

practices > practice3 > gcd.pl

```
1   gcd(A,B,G) :- A = B;B = 0, G = A.
2   gcd(A,B,G) :- A = 0, G = B.
3   gcd(A,B,G) :- A > B, C is A mod B, gcd(C,B,G).
4   gcd(A,B,G) :- A < B, C is B mod A, gcd(C,A,G).
5
```

PROBLEMS OUTPUT TERMINAL ···

```
ices/practice3/gcd.pl'].
true.

?- gcd(12, 4, G).
G = 4 .

?- gcd(12, 6, G).
G = 6 .

?-
```

- You can use the shortcut Alt + X, then L to load Prolog facts and rules in the file.
- Then you can ask queries in Terminal window.

Summary

- PL Paradigm
- Imperative vs. Declarative: Examples.
- Preparing Practices.