GraalPy Quick Reference v1

GraalPy 23.1.0

Designed for use with GraalVM for JDK 21

INSTALL

Downloading (Oracle GraalPy, GraalPy Community)

1. Navigate to GitHub releases and download GraalPy.
2. Uncompress the archive:
   - If you are using macOS Catalina or later, first remove the quarantine attribute:
     ```
     sudo xattr -r -d com.apple.quarantine
     <archive>.tar.gz
     ```

   -
     ```
     tar -xzf <archive>.tar.gz
     ```

Using pyenv

Install GraalPy using Pyenv and specify the GraalPy version:

```
pyenv install graalpy-23.1
```

Using Conda-Forge (GraalPy Community)

Install the latest GraalPy using Conda-Forge:

```
conda create -c conda-forge -n graalpy graalpy
```

You can also build GraalPy from source on Linux, macOS, and Windows

SET UP ENVIRONMENT

Create a virtual environment:

```
graalpy -m venv <venv-dir>
```

Activate the environment in a shell session:

```
source <venv-dir>/bin/activate
```

SET UP DEVELOPMENT

We recommend PyCharm. Create or open a Python project, then create a new virtual environment.

RUN

Use the `graalpy` launcher to run your Python application:

```
graalpy <options>  <-c cmd | filename>
```

Create standalone binaries from Python applications:

```
graalpy -m standalone native \
        --output my_application \
        --module my_python_script.py \
        --venv <venv-dir>
```

INSTALL PACKAGES

Use `pip` to directly install a package or via a *requirements.txt* file. For example:

```
pip install numpy torch
```

Check if your package is compatible with GraalPy at:

https://www.graalvm.org/compatibility/

EMBED IN JAVA

GraalPy Standalone Tool

GraalPy provides a shortcut to create a Maven project skeleton with Python embedded:

1. Create a Java project:
   ```
   graalpy -m standalone polyglot_app \
       --output-directory MyPythonJavaEmbedding
   ```
2. Package and run:
   ```
   mvn exec:exec
   ```

Manual Configuration

To embed Python in an existing Java application, add GraalPy as a Maven or Gradle build tool dependency or explicitly put the JAR file on the module path (requires GraalVM JDK to run).

Maven configuration:

```
<dependency>
  <groupId>org.graalvm.polyglot</groupId>
  <artifactId>polyglot</artifactId>
  <version>23.1.0</version>
</dependency>
<dependency>
  <groupId>org.graalvm.polyglot</groupId>
  <artifactId>python</artifactId>
  <version>23.1.0</version>
  <type>pom</type>
</dependency>
```

Gradle configuration:

```
dependencies {
  implementation("org.graalvm.polyglot:polyglot:23.1.0")
  implementation("org.graalvm.polyglot:python:23.1.0")
  testImplementation("junit:junit:4.13.2")
}
```

DEBUG

Debug Python code with the Chrome Inspector:

```
graalpy --inspect your_script.py
```

GraalPy also works with PyCharm and Python debuggers like `pdb`.

OPTIONS SPECIFIC TO GRAALPY

GraalPy uses a garbage collector that reserves memory based on the amount of total system memory. Optimize GC using Java GC options. For example, restrict GraalPy to use a maximum of one gigabyte of object memory:

```
--vm.Xmx1G
```

Disable JIT compilation for short-running Python scripts:

```
--experimental-options --engine.Compilation=false
```

More info at: graalvm.org