Part I

Problem 0: Install and set up

Read and complete the [**Getting Started guide**](http://web.mit.edu/6.005/www/fa16/getting-started/). The guide will step through:

* installing the JDK, Eclipse
* configuring Eclipse

You need to complete all the steps in the guide before you start working on this problem set.

257/5000

第一部分  
问题0：安装和设置  
阅读并完成入门指南。 指南将逐步完成：  
•安装JDK，Eclipse  
•配置Eclipse  
在开始处理此问题集之前，您需要完成指南中的所有步骤。

Problem 1: Clone and import

Create Java project, and import the codes to your project.

Problem 2: Warm up with may­Use­Code­In­Assignment

1. Look at the source code contained in RulesOf6005.java in package rules. Your warm-up task is to implement:
2. mayUseCodeInAssignment(
3. boolean writtenByYourself, boolean availableToOthers,
4. boolean writtenAsCourseWork, boolean citingYourSource,

boolean implementationRequired)

You can find the policy under [General Information](http://web.mit.edu/6.005/www/fa16/general/) on the [course home page](http://web.mit.edu/6.005/www/fa16/).

1. Once you’ve implemented this method, run the main method in RulesOf6005.java.

public static void main(String[] args) is the entry point for Java programs. In this case, the main method calls the mayUseCodeInAssignment method with input parameters. To run main in RulesOf6005, right click on the file RulesOf6005.java in either your Package Explorer, Project View, or Navigator View, go to the *Run As* option, and click on *Java Application*.

问题1：克隆并导入

创建Java项目，并将代码导入项目。

问题2：使用may-UseCodeInIn-Assignment进行预热

一个。 查看包规则中RulesOf6005.java中包含的源代码。 你的热身任务是实施：

湾mayUseCodeInAssignment（

C。 boolean writtenByYourself，boolean availableToOthers，

d。 boolean writtenAsCourseWork，boolean citingYourSource，

        boolean implementationRequired）

您可以在课程主页的常规信息下找到该政策。

即 实现此方法后，运行RulesOf6005.java中的main方法。

public static void main（String [] args）是Java程序的入口点。 在这种情况下，main方法使用输入参数调用mayUseCodeInAssignment方法。 要在RulesOf6005中运行main，请在Package Explorer，Project View或Navigator View中右键单击RulesOf6005.java文件，转到Run As选项，然后单击Java Application。

Unit testing

Right now, we can use the main method plus some visual inspection to verify that our implementation is correct. More generally, programs will have many dozens of methods that need to be tested; visually inspecting output for each one is fragile, time-consuming, and inherently non-scalable.

Instead, we will use *automated unit testing*, which runs a suite of tests to automatically test whether the implementations are correct. For this problem set, we will write unit tests for methods that do not draw graphics on the screen; unit-testing GUIs is a more complex problem.

Automated unit testing with JUnit

[JUnit](http://www.junit.org/) is a widely-adopted Java unit testing library, and we will use it heavily in 6.005. A major component of the 6.005 design philosophy is to decompose problems into minimal, orthogonal units, which can be assembled into the larger modules that form the finished program. One benefit of this approach is that each unit can be tested thoroughly, independently of others, so that faults can be quickly isolated and corrected as code is rewritten and modules are configured. Unit testing is the technique of writing tests for the smallest testable pieces of functionality, to allow for the flexible and organic evolution of complex, correct systems.

By writing thoughtful unit tests, it is possible to verify the correctness of one’s code, and to be confident that the resulting programs behave as expected. In 6.005, we will use JUnit version 4.

Anatomy of JUnit

JUnit unit tests are written method by method. There is nothing special a class has to do to be used by JUnit; it only need contain methods that JUnit knows to call, which we call *test methods*. Test methods are specified using *annotations*, which may be thought of as keywords (more specifically, they are a type of metadata), that can be attached to individual methods and classes. Though they do not themselves change the meaning of a Java program, at compile- or run-time other code can detect the annotations and make decisions accordingly. Though we will not deeply explore annotations in 6.005, you will see a few important uses of them.

Look closely at RulesOf6005Test.java, and note the @Test that precedes method definitions. This is an example of an annotation. The JUnit library uses this annotation to determine which methods to call when running unit tests. The @Test annotation denotes a test method; there can be any number in a single class. Even if one test method fails, the others will be run.

Unit test methods can contain calls to assertEquals, which is an assertion that compares two objects against each other and fails if they are not equal, assertTrue, which checks if the condition is true, and assertFalse, which checks if the condition is false. [Here is a list of all the assertions supported by JUnit](http://junit.org/junit4/javadoc/latest/org/junit/Assert.html). If an assertion in a test method fails, that test method returns immediately, and JUnit records a failure for that test.

1. Run the unit tests.

To run the tests in RulesOf6005Test, right click on the RulesOf6005Test.java file in either your Package Explorer, Project View, or Navigator View, and go to the *Run As* option. Click on *JUnit Test*, and you should see the JUnit view appear.

If your implementation of mayUseCodeInAssignment is correct, you should see a green bar, indicating that all the tests (there’s only 1 test, containing 2 assertions) passed.

1. Try *breaking* your implementation and running RulesOf6005Test again.

You should see a red bar in the JUnit view, and if you click on test­May­Use­Code­In­Assignment, you will see a *stack trace* in the bottom box, which provides a brief explanation of what went wrong. Double-clicking on a line in the failure stack trace will bring up the code for that frame in the trace. This is most useful for lines that correspond to your code; this stack trace will also contain lines for Java libraries or JUnit itself.

1. Enough breaking: fix your implementation so it’s correct again. Make sure the tests pass.

Passing the JUnit tests we provide does **not** necessarily mean that your code is perfect. You need to review the function specifications carefully, and **always write your own JUnit tests** to verify your code.

Problem 3: Commit and push（忽略）

Github内容暂时不做要求

单元测试

现在，我们可以使用main方法加上一些目视检查来验证我们的实现是否正确。更一般地说，程序将有许多需要测试的方法;目视检查每个输出是脆弱的，耗时的，并且固有地不可扩展。

相反，我们将使用自动化单元测试，它运行一套测试来自动测试实现是否正确。对于这个问题集，我们将为不在屏幕上绘制图形的方法编写单元测试;单元测试GUI是一个更复杂的问题。

使用JUnit进行自动单元测试

JUnit是一个广泛采用的Java单元测试库，我们将在6.005中大量使用它。 6.005设计理念的一个主要组成部分是将问题分解为最小的正交单元，这些单元可以组装成形成完成程序的较大模块。这种方法的一个好处是每个单元都可以独立于其他单元进行彻底测试，以便在重写代码和配置模块时快速隔离和纠正故障。单元测试是为最小的可测试功能部件编写测试的技术，以允许复杂，正确的系统的灵活和有机演变。

通过编写周到的单元测试，可以验证一个代码的正确性，并确信生成的程序按预期运行。在6.005中，我们将使用JUnit版本4。

JUnit的剖析

JUnit单元测试是按方法编写的。 JUnit使用类没有什么特别之处;它只需要包含JUnit知道要调用的方法，我们称之为测试方法。测试方法是使用注释指定的，注释可以被认为是关键字（更具体地说，它们是一种元数据），可以附加到单个方法和类。虽然它们本身不会改变Java程序的含义，但在编译或运行时，其他代码可以检测注释并相应地做出决策。虽然我们不会在6.005中深入探讨注释，但您会看到它们的一些重要用途。

仔细查看RulesOf6005Test.java，并注意方法定义之前的@Test。这是注释的示例。 JUnit库使用此批注来确定运行单元测试时要调用的方法。 @Test注释表示测试方法;一个班级中可以有任何数字。即使一种测试方法失败，其他测试方法也会运行。

单元测试方法可以包含对assertEquals的调用，这是一个断言，它将两个对象相互比较，如果它们不相等则失败，assertTrue，它检查条件是否为真; assertFalse，它检查条件是否为假。以下是JUnit支持的所有断言的列表。如果测试方法中的断言失败，则该测试方法立即返回，并且JUnit记录该测试的失败。

C。运行单元测试。

要在RulesOf6005Test中运行测试，请右键单击Package Explorer，Project View或Navigator View中的RulesOf6005Test.java文件，然后转到Run As选项。单击JUnit Test，您将看到JUnit视图出现。

如果你的mayUseCodeInAssignment的实现是正确的，你应该看到一个绿色条，表明所有测试（只有1个测试，包含2个断言）通过。

d。尝试破坏您的实现并再次运行RulesOf6005Test。

您应该在JUnit视图中看到一个红色条，如果您单击test-May¬UseCode-In-Assignment，您将在底部框中看到一个堆栈跟踪，它提供了错误的简要说明。双击故障堆栈跟踪中的一行将显示跟踪中该帧的代码。这对于与您的代码对应的行非常有用;此堆栈跟踪还将包含Java库或JUnit本身的行。

即足够的破解：修复你的实现，使其再次正确。确保测试通过。

通过我们提供的JUnit测试并不一定意味着您的代码是完美的。您需要仔细查看函数规范，并始终编写自己的JUnit测试以验证代码。

问题3：提交和推送（忽略）

Github的内容暂时不做要求

Part II

Turtle graphics and the Logo language

[Logo](http://en.wikipedia.org/wiki/Logo_%28programming_language%29) is a programming language created at MIT that originally was used to move a robot around in space. Turtle graphics, added to the Logo language, allows programmers to issue a series of commands to an on-screen “turtle” that moves, drawing a line as it goes. Turtle graphics have also been added to many different programming languages, [including Python](http://docs.python.org/2/library/turtle.html), where it is part of the standard library.

In the rest of problem set 0, we will be playing with a simple version of turtle graphics for Java that contains a restricted subset of the Logo language:

* forward(units)   
  Moves the turtle in the current direction by *units* pixels, where units is an integer. Following the original Logo convention, the turtle starts out facing up.
* turn(degrees)   
  Rotates the turtle by angle *degrees* to the right (clockwise), where degrees is a double precision floating point number.

You can see the definitions of these commands in Turtle.java.

**Do NOT use any turtle commands other than forward and turn in your code for the following methods.**

Problem 4: drawSquare

Look at the source code contained in TurtleSoup.java in package turtle.

Your task is to implement drawSquare(Turtle turtle, int sideLength), using the two methods introduced above: forward and turn.

Once you’ve implemented the method, run the main method in TurtleSoup.java. The main method in this case simply creates a new turtle, calls your drawSquare method, and instructs the turtle to draw. Run the method by going to *Run → Run As… → Java Application*. A window will pop up, and, once you click the “Run!” button, you should see a square drawn on the canvas.

Problems 5—10: Polygons and headings

**For detailed requirements, read the specifications of each function to be implemented above its declaration in TurtleSoup.java. Be careful when dealing with mixed integer and floating point calculations.**

You should not change any of the *method declarations* ([what’s a declaration?](http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html)) below. If you do so, you risk receiving **zero points** on the problem set.

Drawing polygons

* Implement calculateRegularPolygonAngle   
  There’s a simple formula for what the inside angles of a regular polygon should be; try to derive it before googling/binging/duckduckgoing.
* Run the JUnit tests in TurtleSoupTest   
  The method that tests calculateRegularPolygonAngle should now pass and show green instead of red.

If testAssertionsEnabled fails, you did not follow the instructions in the Getting Started guide. [Getting Started step 2](http://web.mit.edu/6.005/www/fa16/getting-started/#config-eclipse) has setup you must perform before using Eclipse.

* Implement calculatePolygonSidesFromAngle   
  This does the inverse of the last function; again, use the simple formula. However, **make sure you correctly round** to the nearest integer. Instead of implementing your own rounding, look at Java’s [java.lang.Math](http://docs.oracle.com/javase/8/docs/api/?java/lang/Math.html) class for the proper function to use.
* Implement drawRegularPolygon   
  Use your implementation of calculateRegularPolygonAngle. To test this, change the main method to call drawRegularPolygon and verify that you see what you expect.

Calculating headings

* Implement calculateHeadingToPoint   
  This function calculates the parameter to turn required to get from a current point to a target point, with the current direction as an additional parameter. For example, if the turtle is at (0,1) facing 30 degrees, and must get to (0,0), it must turn an additional 150 degrees, so calculateHeadingToPoint(30, 0, 1, 0, 0) would return 150.0.
* Implement calculateHeadings   
  Make sure to use your calculateHeadingToPoint implementation here. For information on how to use Java’s List interface and classes implementing it, look up [java.util.List](http://docs.oracle.com/javase/8/docs/api/?java/util/List.html) in the Java library documentation. Note that for a list of *n* points, you will return *n-1* heading adjustments; this list of adjustments could be used to guide the turtle to each point in the list. For example, if the input lists consisted of xCoords=[0,0,1,1] and yCoords=[1,0,0,1] (representing points (0,1), (0,0), (1,0), and (1,1)), the returned list would consist of [180.0, 270.0, 270.0].

Problem 11: Personal art

* Implement drawPersonalArt   
  In this function, you have the freedom to draw any piece of art you wish. Your work will be judged both on aesthetics and on the code used to draw it. Your art doesn’t need to be complex, but it should be more than a few lines. Use helper methods, loops, etc. rather than simply listing forward and turn commands.

**For drawPersonalArt only, you may also use the color method of Turtle to change the pen color.** You may only use the provided colors.

[Here are some examples](https://www.google.com/search?q=python+turtle+example+images) of the kinds of images you can generate procedurally with turtle graphics, though note that many of them use more commands than what we’ve provided here. Modify the main method to see the results of your function.

第二部分

龟图形和徽标语言

徽标是在麻省理工学院创建的一种编程语言，最初用于在空间中移动机器人。添加到徽标语言中的Turtle图形允许程序员向屏幕上的“乌龟”发出一系列命令，这些“乌龟”会移动，随着它划一条线。 Turtle图形也被添加到许多不同的编程语言中，包括Python，它是标准库的一部分。

在问题集0的其余部分中，我们将使用Java的简单版本的龟图形，其中包含Logo语言的受限子集：

•正向（单位）

以单位像素为单位在当前方向上移动乌龟，其中单位为整数。按照原来的Logo惯例，龟开始面朝上。

•转（度）

将乌龟向右旋转角度（顺时针），其中度数是双精度浮点数。

您可以在Turtle.java中看到这些命令的定义。

请勿使用除转发之外的任何海龟命令，并将代码转入以下方法。

问题4：drawSquare

查看包龟中TurtleSoup.java中包含的源代码。

你的任务是使用上面介绍的两种方法实现drawSquare（Turtle turtle，int sideLength）：forward和turn。

一旦实现了该方法，就在TurtleSoup.java中运行main方法。在这种情况下，主要方法只是创建一个新的乌龟，调用drawSquare方法，并指示乌龟绘制。通过转到Run→Run As ...→Java Application运行该方法。将弹出一个窗口，单击“运行！”按钮后，您将看到画布上绘制的正方形。

问题5-10：多边形和标题

有关详细要求，请阅读要在TurtleSoup.java中声明之上实现的每个函数的规范。处理混合整数和浮点计算时要小心。

你不应该改变下面的任何方法声明（什么是声明？）。如果这样做，则可能会在问题集上收到零点。

绘制多边形

•实现calculateRegularPolygonAngle

对于正多边形的内角应该是一个简单的公式;尝试在googling / binging / duckduckgoing之前派生它。

•在TurtleSoupTest中运行JUnit测试

测试calculateRegularPolygonAngle的方法现在应该通过并显示绿色而不是红色。

如果testAssertionsEnabled失败，则表示您未按照“入门指南”中的说明进行操作。入门步骤2具有在使用Eclipse之前必须执行的设置。

•实现calculatePolygonSidesFromAngle

这与最后一个函数相反;再次，使用简单的公式。但是，请确保正确舍入到最接近的整数。不要实现自己的舍入，而是查看Java的java.lang.Math类以获取正确的函数。

•实现drawRegularPolygon

使用calculateRegularPolygonAngle的实现。要对此进行测试，请更改main方法以调用drawRegularPolygon并验证您是否看到了预期结果。

计算标题

•实现calculateHeadingToPoint

此函数计算从当前点到目标点所需的参数，将当前方向作为附加参数。例如，如果乌龟在（0,1）面向30度，并且必须到达（0,0），它必须再转150度，因此calculateHeadingToPoint（30,0,1,0,0）将返回150.0。

•实施calculateHeadings

确保在此处使用calculateHeadingToPoint实现。有关如何使用Java的List接口和实现它的类的信息，请在Java库文档中查找java.util.List。请注意，对于n个点的列表，您将返回n-1个标题调整;此调整列表可用于指导乌龟到列表中的每个点。例如，如果输入列表由xCoords = [0,0,1,1]和yCoords = [1,0,0,1]组成（表示点（0,1），（0,0），（1， 0）和（1,1）），返回的列表将包含[180.0,270.0,270.0]。

问题11：个人艺术

•实现drawPersonalArt

在此功能中，您可以自由地绘制任何您想要的艺术品。您的工作将根据美学和用于绘制它的代码来判断。你的艺术不需要复杂，但它应该不仅仅是几行。使用辅助方法，循环等，而不是简单地列出forward和turn命令。

仅对于drawPersonalArt，您还可以使用Turtle的颜色方法来更改笔颜色。您只能使用提供的颜色。

以下是您可以使用龟图形程序生成的图像种类的一些示例，但请注意，其中许多图像使用的命令多于我们在此处提供的命令。修改main方法以查看函数的结果。

可参考原网页：http://web.mit.edu/6.005/www/fa16/psets/ps0/