

APS 106 - FUNDAMENTALS OF COMPUTER PROGRAMMING

LAB ASSIGNMENT # 1

For this and all subsequent lab assignments, the following guidelines apply:

1. Students must work individually, and **each student must submit a program which is his/her own work**. You may consult your friends and the TAs during the labs; however, you are NOT working in groups. Do not share files with others. The labs offer a chance to learn to program: take advantage of that opportunity, and make sure your friends take advantage of it too.
2. All labs will be posted on the course website, and you must do the labs in your assigned computer lab timeslot where a TA will be present to help you learn. Python and the Wing101 development environment are already installed on the computers in the Engineering Computing Facility (ECF) labs. Alternately, you can follow the instructions posted on the APS106 Quercus site to install Python and the Wing101 development environment on your own computer (both programs are freely available to download from the Internet, with installation instructions below).
3. You need to do lab preparation for each lab excluding the first lab. Preparation needs to be done on an engineering notebook, and it will be marked by a TA at the beginning of your lab session. You are required to bring this engineering notebook to every lab including the first lab. **There is no lab preparation for the first lab. You will get 3 points if you bring your engineering notebook to the first lab.** For all the remaining labs, you will obtain 3 points if you complete the lab preparation (see lab preparation instructions on Quercus). You can obtain an engineering notebook from UofT bookstore or the engineering bookstore.
4. For the first lab, you will need to call a TA at the last 30 min of your lab session, and show him/her your work for all parts. He/she will give you a mark out of 10 according to the marking scheme. Make sure you save enough time at the end for the TAs to mark your work. The TAs will not stay for marking after the lab ends.

OBJECTIVE OF THIS LAB – to introduce you to Python programming with Wing101, and show you how you can use the software in a simple way. There are three parts in this lab which will focus on identifying and debugging errors and modifying an existing program. This won't take you very long. Labs in the coming weeks will be more challenging and extensive.

LAB 1 MARK BREAKDOWN (OUT OF 10 POINTS)

Lab preparation - Bring engineering notebook – 3 points

Part 1: Debugging - Identify errors and correctly draw part2 shape – 3 points

Part 2: New shape - Correctly draw part3 shape – 3 points

Part 3: Submission - Submit completed code to Markus – 1 point

INTRODUCTION – For this first lab assignment, we will be using a drawing tool, called **Turtle**. Turtle is a part of the Python programming language and allows us to create drawings from our program. The idea is that the turtle runs around on the screen and the turtle's tail can be up or down. When it is down, the turtle leaves a trail and draws on the screen as it moves. In our case, the turtle is called **alex** and the initialization code to bring the turtle to life (show up in the center of the screen as a little triangle) has already been written for you. Your task is to issue the correct commands to control **alex** the turtle, such that it draws the right things. The turtle understands the following commands:

Command	Action
<code>alex.up()</code>	Lift the tail (stop drawing)
<code>alex.down()</code>	Lower the tail (start drawing)
<code>alex.right(d)</code>	Turn right by d degrees
<code>alex.left(d)</code>	Turn left by d degrees
<code>alex.forward(s)</code>	Moves steps in the current direction
<code>alex.backward(s)</code>	Moves steps backwards with the current heading
<code>alex.setheading(d)</code>	Change heading to direction d (0: east, 90: north, 180: west, 270: south)
<code>alex.goto(x, y)</code>	Move to coordinates (x, y)
<code>alex.circle(r, d)</code>	Move in circle with r radius for d degrees (counterclockwise if $d > 0$)

PART 1: DEBUGGING AND IDENTIFY ERRORS

Programming is a complex process, and because human beings are not perfect, they often make mistakes. Therefore, it is very common for a program to contain errors. Programming errors are called bugs and the process of tracking them down and correcting them is called *debugging*.

There are many types of errors that can occur in a program, for the purposes of this lab we will focus on four kinds: syntax errors, runtime errors, semantic errors and logical errors. It is useful to distinguish between them in order to track them down more quickly.

Syntax errors

The compiler can only compile a program if the program is syntactically correct; otherwise, the process fails and returns an error message. Syntax refers to the structure of a program and the rules about that structure. For example, in English, a sentence must begin with a capital letter and end with a period. this sentence contains a syntax error. So does this one

For most readers, a few syntax errors are not a significant problem, which is why we can read the poetry of E. E. Cummings without spewing error messages. Python is not so forgiving. If there is a single syntax error anywhere in your program, Python will print an error message and quit, and you will not be able to run your program. During the first few weeks of your programming career, you will probably spend a lot of time tracking down syntax errors. As you gain experience, though, you will make fewer errors and find them faster.

Runtime errors

The second type of error is a runtime error, so called because the error does not appear until you run the program. These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.

Runtime errors are rare in the simple programs you will see in the beginning, so it might be a while before you encounter one.

Semantic errors

The third type of error is the semantic error which results from improper use of the statements or variables. In this case the syntax of the programming language is correct but the problem could result from the use of wrong variables, wrong operations or in the wrong order.

Semantic errors may result from applying an operator not intended for the variable type, calling a function with the wrong number of arguments or with arguments of the wrong type, etc.

Logical errors

The fourth type of error is the logical error. A logical error is when the wrong output is produced due to a miscalculation or misunderstanding of the requirements. These type of errors are generally the most difficult to fix because the code will execute without crashing. There are no error messages produced.

Identifying logical errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing. Another method of debugging logical errors is to step through the program one instruction at a time to figure out where things go wrong.


Let's get some practice in debugging!

Debugging

You are to complete all portions of these instructions, including submission of the modified program in step 12, by the end of your lab period. In subsequent weeks, you will be required to upload your files to MarkUs.

1. On the APS106 Lab Quercus site under "Assignments", find "Lab 1".
2. On your ECF desktop, click on the Computer icon and find My Documents (W:). On the W: drive, create a folder called APS106, in which you'll keep all of your labs for the course. Inside APS106, create a folder called lab1, for the assignment this week.
3. Download the file *lab1.py* and the file *Windows_Python_Wing101.pdf* from Quercus.
4. Become familiar with the Wing101 environment.
 - ☐ If you are working on an ECF computer, then Python and Wing101 are already installed, so start reading at **Step 3: Running Wing101** (page 9) of the document and complete all the exercises Step 3 until the end of the document.
 - ☐ If you are working on a non-ECF computer (for instance, your own Windows computer), then start at **Step 1: Install Python** and complete all the exercises in the document. There is an alternative set of instructions for Mac users in the file *OSX_Python_Wing101.pdf*, also available on the course website.
5. Now that you've made your first program by yourself, we want you to explore some existing code. Right click lab1.py, choose Properties, and where it says Open With, choose Wing101. (You've now made Wing101 the default application for opening Python files.) All

you need to do is double click on lab1.py and Wing101 should open it. Open lab1.py from the lab1 folder.

6. Before you run the program, use comments (comment with #) to add your name and section number to the second and third line of the program.
7. *lab1.py* uses the turtle package to generate graphics. This is not the only graphics package, but it is a relatively simple one to start with. Read through the in-line comments that describe what *lab1.py* does. Notice how clearly documenting code improved the readability of the code and increase the ability of a new coder to understand code written by someone else.
8. Try to visualize what the code is doing following through each step.
9. You can run the code by clicking on  or holding down ctrl+alt+v (in Wing101). This code has some errors in it, so you will need to make all the corrections before the turtle will draw out the shape below for you. As you are making the changes be sure to write them down in the table provided below. Hint: Is there anything inconsistent with the comments? Is it really doing what it says?

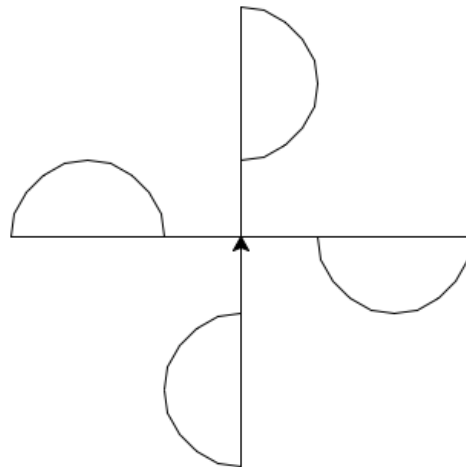
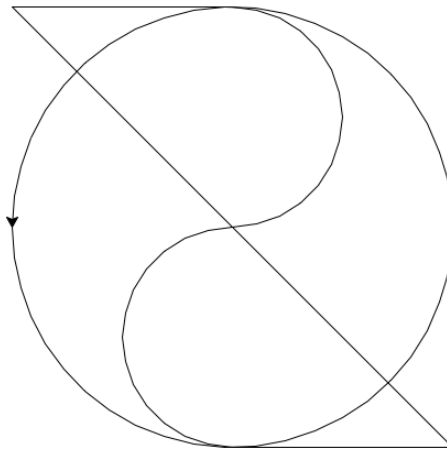


Table 1. List of Errors in lab1.py

Line Number	Error Description	Type of error (Syntax, Runtime, Semantic, or Logical)

Part 2: Draw New Shape

10. In this part, you are supposed to comment out code of part 1, and utilize and modify some lines to have the turtle draw a shape that looks exactly like the figure shown below:



Part 3: Submission

11. Save your program to a file named **lab1.py** and submit it on MarkUS (<https://markus.engineering.utoronto.ca/aps106>). Login with your UTORID. More information is at the top of the 'Assignments' page on Quercus under 'Lab Submission Instructions'.

12. Once you have completed all the tasks, you need to call up one of the TAs and demonstrate the working program with all the errors removed. The TA will then provide you with a mark out of 10 depending on how well you have completed the lab requirements.

Note: The lab1.py program and lab1 assignment were adapted from Robert Hesse, 2014.