

APS 106 - FUNDAMENTALS OF COMPUTER PROGRAMMING

LAB ASSIGNMENT # 1

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

1. You must work individually, and **you must submit a program which is your 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. You may work on the labs during the scheduled practical sessions where a TA will be available to help you learn OR on your own outside of practical sessions. You are free to attend any of the weekly practical sessions; however, priority for TA time will go to students attending their registered section. 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).

OBJECTIVE OF THIS LAB – to introduce you to Python programming with Wing101 and to submitting your code to MarkUs. There are two parts in this lab which will focus on identifying and debugging errors and modifying an existing program. As the term progresses, labs will be more challenging and extensive.

WHAT YOU MUST SUBMIT TO MARKUS FOR THIS LAB:

1. A python file saved as “lab1_p1.py” for part 1
2. A python file saved as “lab1_p2.py” for part 2

Please review the “Introduction to MARKUS” video and “MARKUS submission checklist” on quercus before submitting. Submissions failing autograder tests that do not adhere to the checklist will **NOT** be regraded.

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>	Move s steps in the current direction
<code>alex.backward(s)</code>	Move s 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:

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*. Learning how to find and correct bugs your programs will be essential skill in completing the remaining labs.

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 types 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!

Part 1 - Debugging

You are to complete all portions of these instructions, including submission of the modified program in step 13.

- 1) On the APS106 Lab Quercus site under "Lab Materials", find "Lab 1".

Note: The lab1_p1.py program and lab1 assignment were adapted from Robert Hesse, 2014.

- 2) On your computer, 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_p1.py* and the file *Windows_Python_Wing101.pdf* from Quercus into your lab1 folder.
- 5) Become familiar with the Wing101 environment by opening *Windows_Python_Wing101.pdf* if you are using a windows machine or *OSX_Python_Wing101.pdf* if you are using a mac. Start at **Step 1: Install Python** and complete all the exercises in the document.
- 6) Now that you've made your first program by yourself, we want you to explore some existing code. Right click *lab1_p1.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_p1.py* and Wing101 should open it. Open *lab1_p1.py* from the lab1 folder.
- 7) 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. It is a good idea to do this for all your labs in this course.
- 8) *lab1_p1.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 inline comments that describe what *lab1_p1.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.
- 9) Try to visualize what the code is doing following through each step.
- 10) 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?

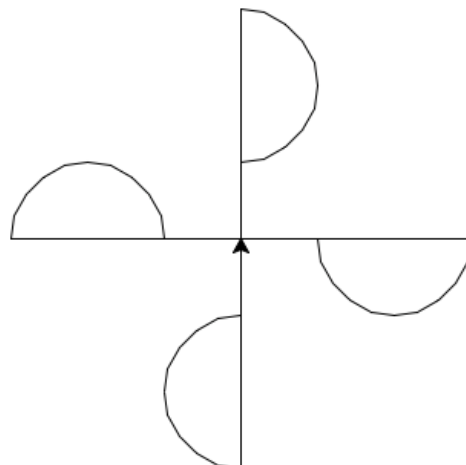


Table 1. List of Errors in lab1.py

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

11) Once you have corrected all the bugs in the program, submit the lab1_p1.py file to MarkUs. Your submission for this part will be graded according to the following:

- a) The program is correctly submitted to MarkUs [1 mark]
- b) The program executes without any errors [2 marks]
- c) The program creates the correct image using the turtle [2 marks]

To check if your program is correctly submitted to MarkUs, execute the automated tests and check the results. To see how to do this, see the 'MarkUs How To' video posted to quercus.

Part 2:

In this part of the lab you will write a program that draws your initials (first and last) using the turtle module.

- 1) Download the file lab1_p2.py from quercus and open in Wing101. We have written some of the code for you. Read through the code and the comments to familiarize yourself with the code provided.
- 2) Add lines of code to make alex the turtle draw the initials of your first and last name. You may choose to use lower or uppercase letters. You can choose to style the letters in the format of your choice (e.g. block, bubble, cursive, etc.). The only requirement is that both letters must be clearly legible.
- 3) When finished, submit your code to MarkUs. Your submission will be graded according to the following:
 - a) The program is correctly submitted to MarkUs [1 mark]
 - b) The program executes without any errors [2 marks]
 - c) The program creates an image of your initials using the turtle [2 marks]