PYTHON BOOT CAMP

Module 1: An Introduction

Objectives

- ☞ To understand computer basics, programs, and operating systems (§§1.2-1.4).
- ☞ To write and run a simple Python program (§1.5).
- ☞ To explain the basic syntax of a Python program (§1.5).
- ☞ To describe the history of Python (§1.6).
- ☞ To explain the importance of, and provide examples of, proper programming style and documentation (§1.7).
- ☞ To explain the differences between syntax errors, runtime errors, and logic errors (§1.8).
- ☞ To create a basic graphics program using Turtle (§1.9).

Programs

Computer *programs*, known as *software*, are instructions to the computer.

You tell a computer what to do through programs. Without programs, a computer is an empty machine. Computers do not understand human languages, so you need to use computer languages to communicate with them.

Programs are written using programming languages.

Programming Languages

Machine Language   Assembly Language   High-Level Language

Machine language is a set of primitive instructions built into every computer. The instructions are in the form of binary code, so you have to enter binary codes for various instructions. Program with native machine language is a tedious process. Moreover the programs are highly difficult to read and modify. For example, to add two numbers, you might write an instruction in binary like this:

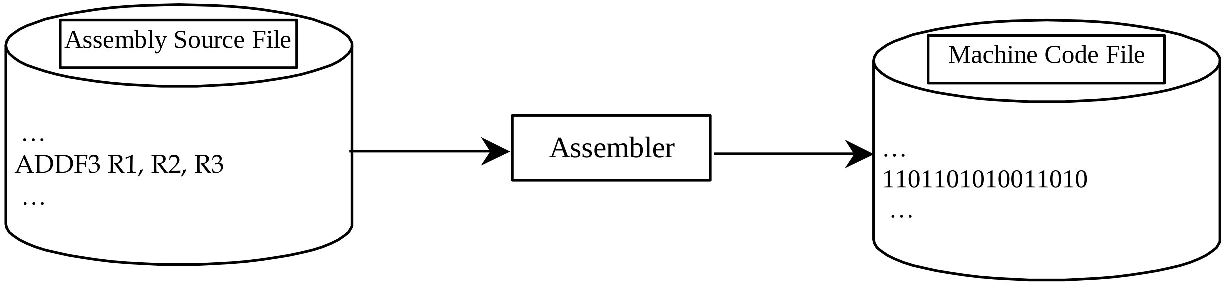
```
1101101010011010
```

Programming Languages

Machine Language Assembly Language High-Level Language

Assembly languages were developed to make programming easy. Since the computer cannot understand assembly language, however, a program called assembler is used to convert assembly language programs into machine code. For example, to add two numbers, you might write an instruction in assembly code like this:

```
ADDF3 R1, R2, R3
```



Programming Languages

Machine Language   Assembly Language   High-Level Language

The high-level languages are English-like and easy to learn and program. For example, the following is a high-level language statement that computes the area of a circle with radius 5:

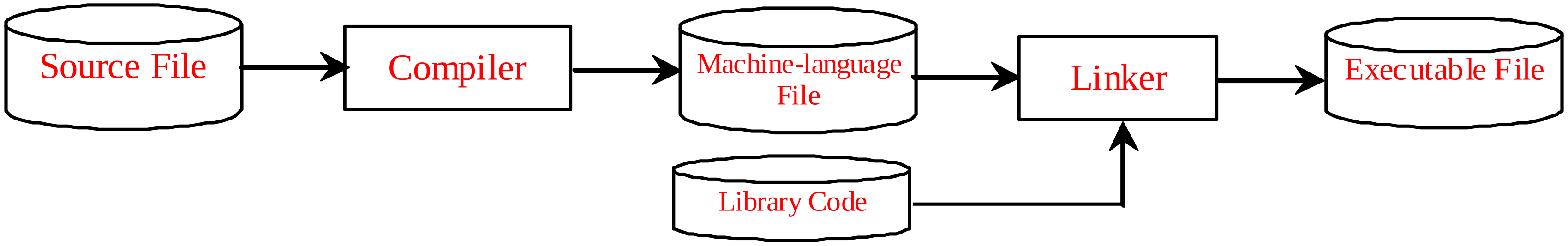
```
area = 5 * 5 * 3.1415;
```

Popular High-Level Languages

- COBOL (COmmon Business Oriented Language)
- FORTRAN (FORmula TRANslation)
- BASIC (Beginner All-purpose Symbolic Instructional Code)
- Pascal (named for Blaise Pascal)
- Ada (named for Ada Lovelace)
- C (whose developer designed B first)
- Visual Basic (Basic-like visual language developed by Microsoft)
- Delphi (Pascal-like visual language developed by Borland)
- C++ (an object-oriented language, based on C)
- C# (a Python-like language developed by Microsoft)
- Python (We use it in the book)

Compiling Source Code

A program written in a high-level language is called a *source program*. Since a computer cannot understand a source program. Program called a *compiler* is used to translate the source program into a machine language program called an *object program*. The object program is often then linked with other supporting library code before the object can be executed on the machine.



What is Python?

General Purpose   Interpreted   Object-Oriented

Python is a general purpose programming language. That means you can use Python to write code for any programming tasks. Python are now used in Google search engine, in mission critical projects in NASA, in processing financial transactions at New York Stock Exchange.

What is Python?

General Purpose Interpreted Object-Oriented

Python is interpreted, which means that python code is translated and executed by an interpreter one statement at a time.

In a compiled language (not Python), the entire source code is compiled and then executed altogether.

What is Python?

General Purpose Interpreted Object-Oriented

Python is an object-oriented programming language. Data in Python are objects created from classes. A class is essentially a type that defines the objects of the same kind with properties and methods for manipulating objects. Object-oriented programming is a powerful tool for developing reusable software.

Python’s History

- created by Guido van Rossum in Netherlands in 1990
- Open source

Python 2 vs. Python 3

Python 3 is a newer version, but it is not backward compatible with Python 2. That means if you write a program using Python 2, it may not work on Python 3.

Installing Python

- Installing Python is (usually) a two-step process:
  - Install Python itself
    - What this does is give your computer the ability to understand the Python code you will type.
    - It will also allow your computer to “compile”, or interpret, the Python code you will type
  - Install some type of Integrated Development Enviroment (IDE)

Installing Python

- So what is an IDE?
  - Let me ask you a question: What is Microsoft Word?
    - "Well, it's a program to type…stuff…reports, resumes, etc."
  - What is Excel?
    - It's a spreadsheet program
  - Well, just like you have a program to type reports (Word) or a program for spreadsheets (Excel), you also have a specific program for coding!
  - And this program is known as an IDE
  - And there are MANY different IDEs for each language
    - Just like many different spreadsheet programs

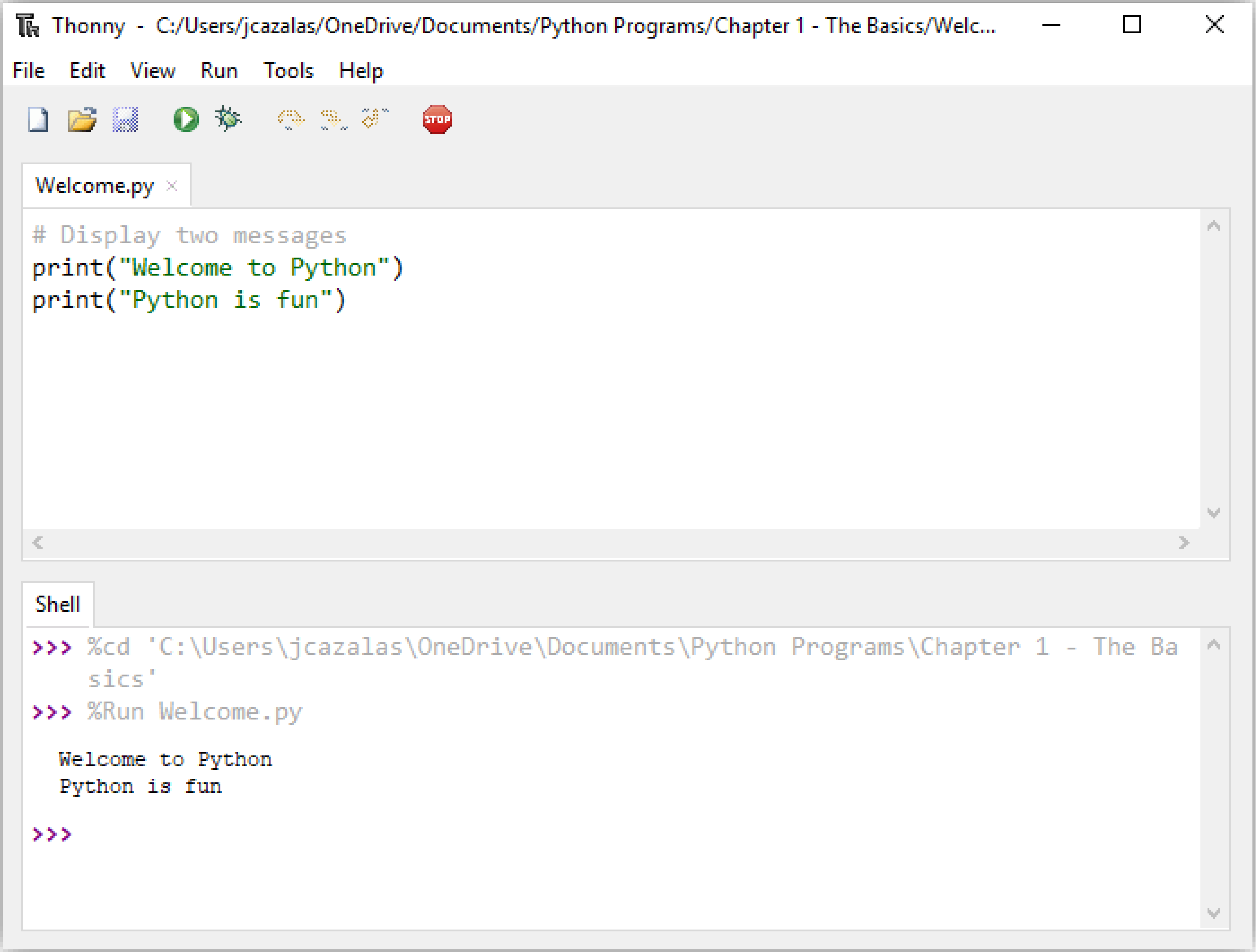
Installing Python

- So while we could install Python and our IDE of choice in a two-step process…
- We'll make it even easier for you:
  - Thonny: www.thonny.org
  - This is a great, beginner IDE
  - And it already has the Python language built in!
  - Meaning, you needn't perform the first step of downloading and installing Python
  - So all you do is download and install Thonny

Installing Python

- Go ahead and install Thonny
  - Website: www.thonny.org
  - At the top right, you will find links for the Windows, Mac, and Linux versions
  - Click whichever is needed for your computer
  - Once downloaded, click the file to install
  - Keep ALL defaults and just click next throughout the install and then Finish once complete

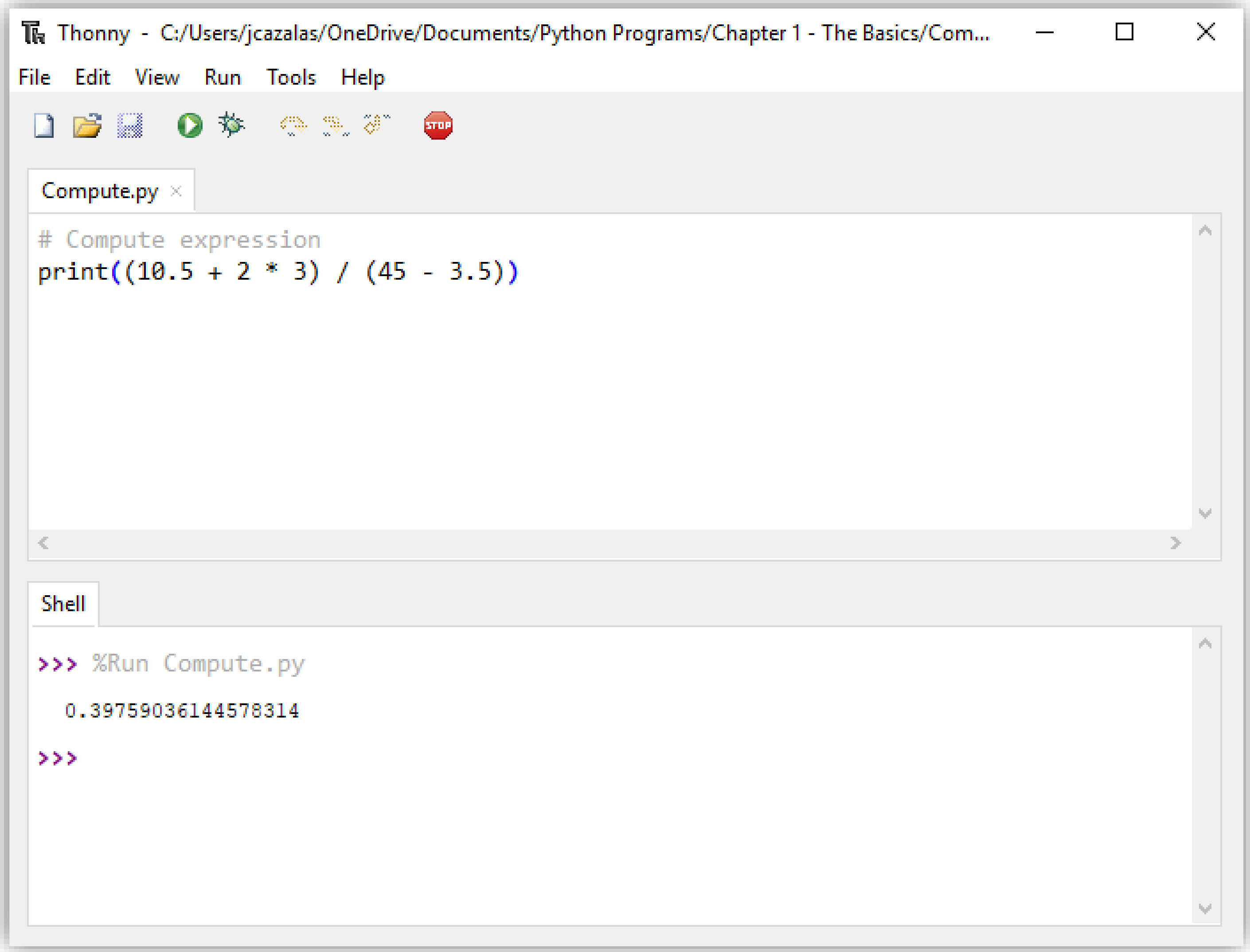
Your First Program


Thonny - C:/Users/jcazalas/OneDrive/Documents/Python Programs/Chapter 1 - The Basics/Welc...
File Edit View Run Tools Help
Welcome.py
Shell

```python
# Display two messages
print("Welcome to Python")
print("Python is fun")
```

```
>>> %cd 'C:\Users\jcazalas\OneDrive\Documents\Python Programs\Chapter 1 - The Ba
    sics'
>>> %Run Welcome.py

  Welcome to Python
  Python is fun

>>>
```

And Another One…


Thonny - C:/Users/jcazalas/OneDrive/Documents/Python Programs/Chapter 1 - The Basics/Com...
File Edit View Run Tools Help
Compute.py
Shell

```
# Compute expression
print((10.5 + 2 * 3) / (45 - 3.5))
```

```
>>> %Run Compute.py

  0.39759036144578314

>>>
```

Anatomy of a Python Program

- Statements
- Comments
- Indentation

Statement

A statement represents an action or a sequence of actions. The statement print("Welcome to Python") in the program in Listing 1.1 is a statement to display the greeting "Welcome to Python“.

```
# Display two messages
print("Welcome to Python")
print("Python is fun")
```

Indentation

The indentation matters in Python. Note that the statements are entered from the first column in the new line. It would cause an error if the program is typed as follows:

```
# Display two messages
 print("Welcome to Python")
print("Python is fun")
```

Special Symbols

Character	Name	Description
()	Opening and closing parentheses	Used with functions.
#	Pound sign	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
''' '''	Opening and closing quotation marks	Enclosing a paragraph comment.

Programming Style and Documentation

- ☞ Appropriate Comments
- ☞ Proper Indentation and Spacing Lines

Appropriate Comments

Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.

Include your name, class section, instructor, date, and a brief description at the beginning of the program.

Proper Indentation and Spacing

- Indentation
  - Indent four spaces.
  - A consistent spacing style makes programs clear and easy to read, debug, and maintain.
- Spacing
  - Use blank line to separate segments of the code.

Programming Errors

- Syntax Errors
  - Error in code construction
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result

What is Computer Science?

- Computer Science can be summarized with two simple words: **problem solving**.
- Computer Science is the study of problems, problem-solving, and the solutions that come out of this problem-solving process.
- Given a problem, the goal is to develop an ***algorithm*** to solve the problem.
- An algorithm is a step-by-step list of instructions to solve the problem.

Algorithm

- Algorithm: a step-by-step series of instructions to complete a task
- Similar to a recipe!
    - a step-by-step series of instructions to prepare a specific food
        - Such as spaghetti!
- So you can think of a recipe as a type of algorithm that is specific for making food.

What is Programming?

- ☞ Once you have developed the algorithm on paper, you must now “prove it” and show that it works.
- ☞ Programming is the process of encoding your algorithm into a programming language, so that it can then be executed by a computer.
- ☞ But what is the first step?
- ☞ You need a solution.
- ☞ This means you need an algorithm!

So who is good at Programming?

- ☞ Are you good at problem solving?
- ☞ Are you good at strategy?
- ☞ These are the core fundamentals of programming.

Program Design & Problem-Solving Techniques

How Do We Write a Program?

- A Computer is not intelligent.
  - It cannot analyze a problem and come up with a solution.
  - A human (the *programmer*) must analyze the problem, develop the instructions for solving the problem, and then have the computer carry out the instructions.
- To write a program for a computer to follow, we must go through a two-phase process: ***problem solving*** and ***implementation.***

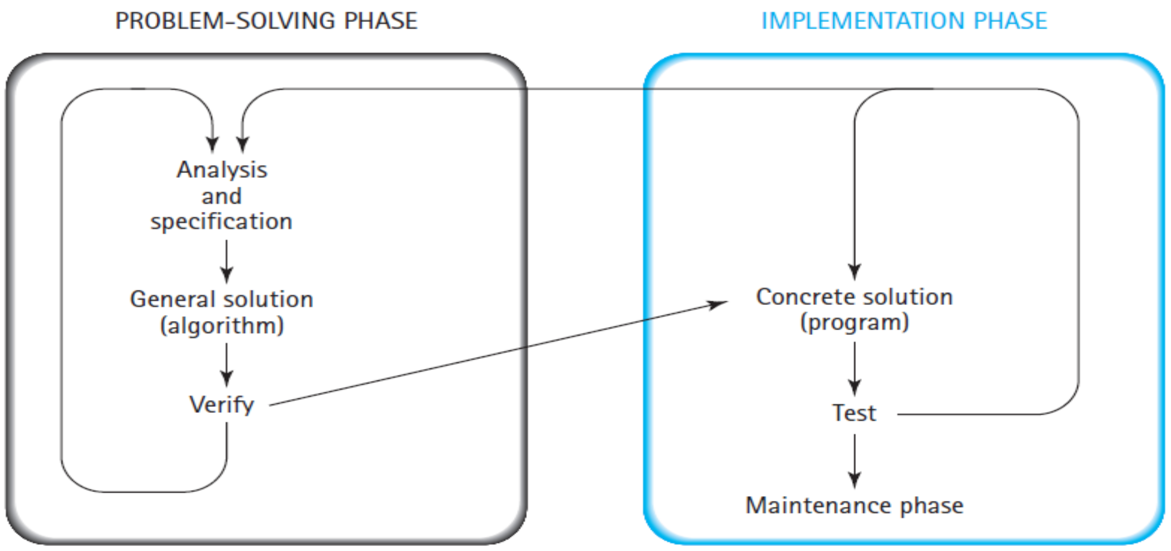

PROBLEM-SOLVING PHASE
IMPLEMENTATION PHASE
Analysis and specification
General solution (algorithm)
Verify
Concrete solution (program)
Test
Maintenance phase

Figure *Programming process*

Problem-Solving Phase (PSP)

- *Analysis and Specification*- Understand (define) the problem and what the solution must do.
- *General Solution (Algorithm)*- Specify the required data types and the logical sequences of steps that solve the problem.
- *Verify*- Follow the steps exactly to see if the solution really does solve the problem.

Implementation Phase

- *Concrete Solution (Program)*- Translate the algorithm (the general solution) into a programming language.
- *Test*- Have the computer follow the instructions.
  - Then manually check the results.
  - If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections.
- Once a program is tested, it enters into next phase (maintenance).
- Maintenance requires Modification of the program to meet changing requirements or to correct any errors that show up while using it.

Weekly Practice Problems

- Repl.it
  - You will have weekly practice problems assigned every Monday
  - These problems will be “due” the by 11:59 p.m. the following Sunday night
  - Due?
    - There is no grade for this Python Bootcamp
    - What you get out of the course directly relates to how dedicated you are and how much you practice
  - A link to enroll in the repl.it course will be posted in Slack

GUI

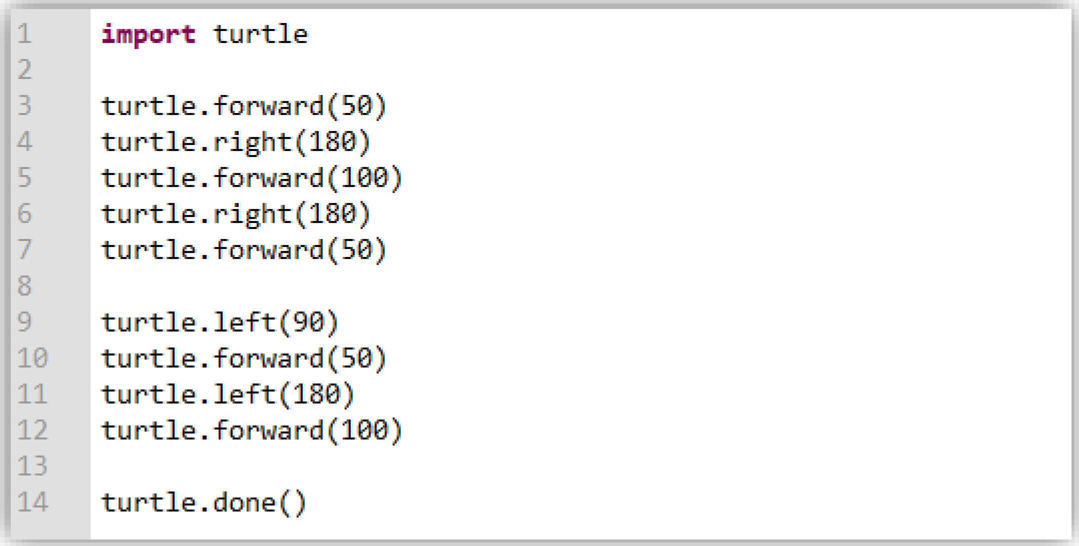
- So what is a GUI?
  - It stands for Graphical User Interface

- We’ll see much about this later in the semester
- But for now, we introduce Python’s turtle!
  - Mainly used to facilitate learning

- Some examples…

GUI

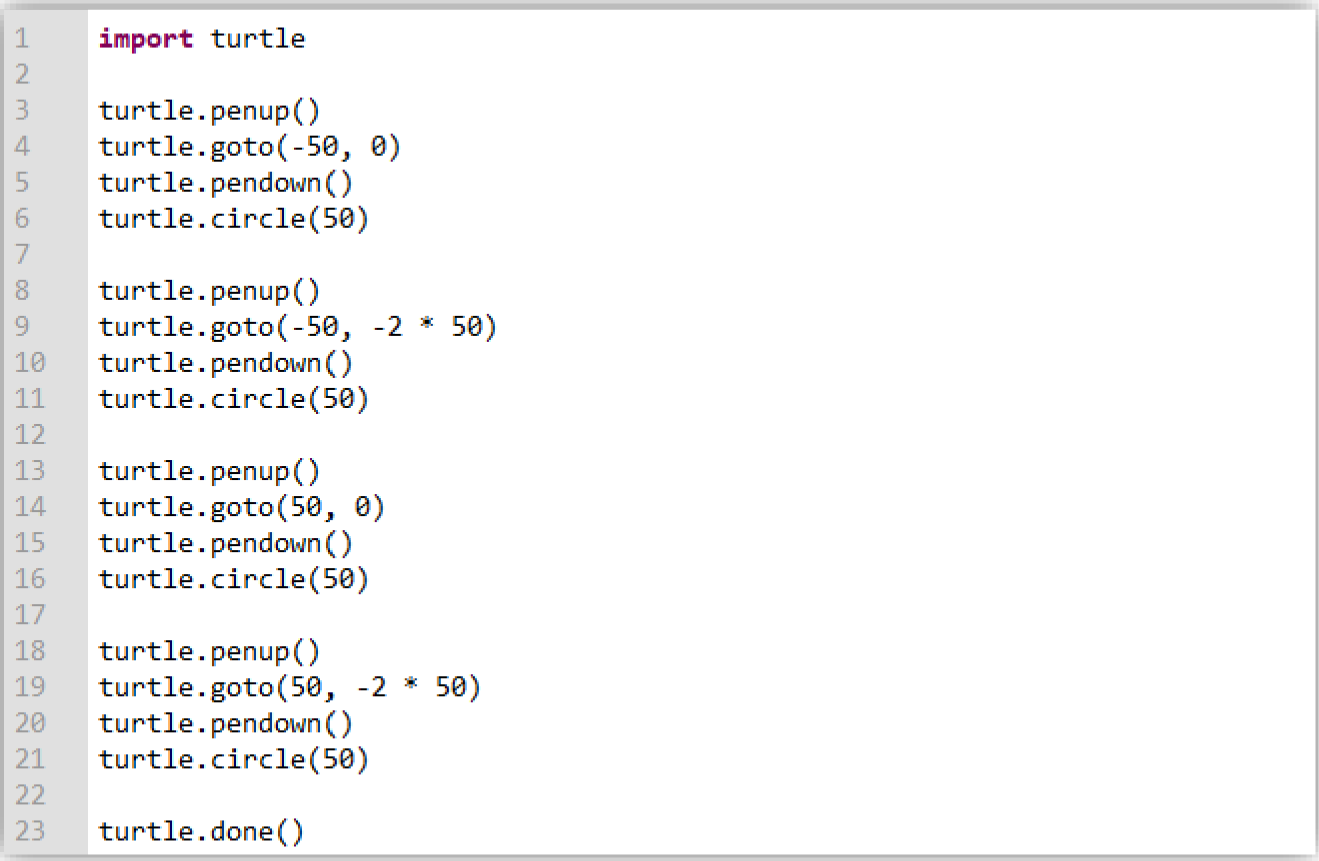
- Turtle Example 1:



```python
import turtle

turtle.forward(50)
turtle.right(180)
turtle.forward(100)
turtle.right(180)
turtle.forward(50)

turtle.left(90)
turtle.forward(50)
turtle.left(180)
turtle.forward(100)

turtle.done()
```

GUI

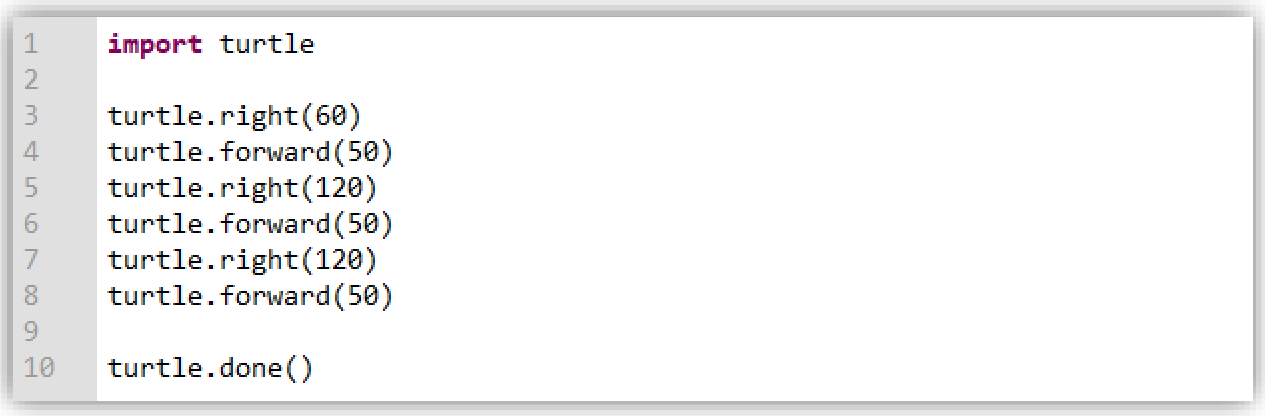
- Turtle Example 2:



```python
import turtle

turtle.penup()
turtle.goto(-50, 0)
turtle.pendown()
turtle.circle(50)

turtle.penup()
turtle.goto(-50, -2 * 50)
turtle.pendown()
turtle.circle(50)

turtle.penup()
turtle.goto(50, 0)
turtle.pendown()
turtle.circle(50)

turtle.penup()
turtle.goto(50, -2 * 50)
turtle.pendown()
turtle.circle(50)

turtle.done()
```

GUI

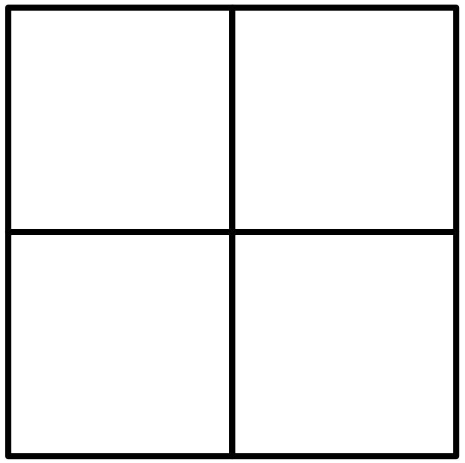
- Turtle Example 3:



```python
import turtle

turtle.right(60)
turtle.forward(50)
turtle.right(120)
turtle.forward(50)
turtle.right(120)
turtle.forward(50)

turtle.done()
```

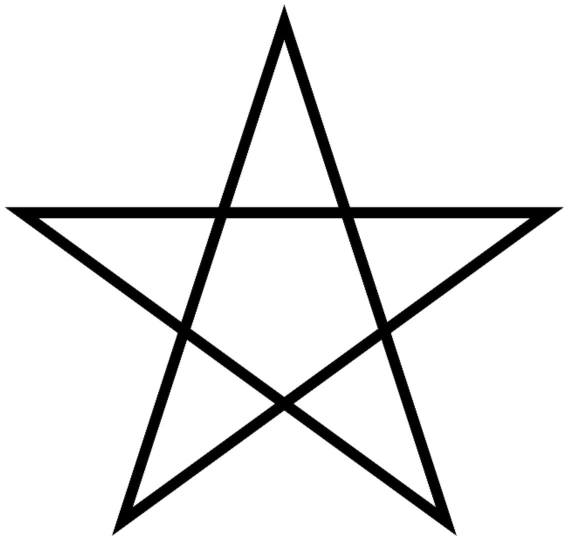
GUI

- Turtle Program --> Your Turn!
  - Make a NEW program and code it such that the following shape is printed on your screen



GUI

- Turtle Program --> Your Turn!
  - Make a NEW program and code it such that the following shape is printed on your screen



  - Hint:
    - You may need to google how large the angles of a star are

PYTHON BOOT CAMP

Module 1:
An Introduction