(venv) (base) harsh@harsh-Inspiron-7570:~/Skillevant_VirtualEnv/Codebase$ python trial.py
/home/harsh/Skillevant_VirtualEnv/venv/lib/python3.7/site-packages/torch/cuda/__init__.py:52: UserWarning: CUDA initialization: Found no NVIDIA driver on your system. Please check that you have an NVIDIA GPU and installed a driver from http://www.nvidia.com/Download/index.aspx (Triggered internally at /pytorch/c10/cuda/CUDAFunctions.cpp:100.)
  return torch._C._cuda_getDeviceCount() > 0
Times,serif
9.96264
Content: 16
Content: CHAPTER 2. WRITING SIMPLE PROGRAMS
Content: The simplest kind of expression is a
Content: you can find the numbers
Content: all examples of numeric literals, and their meaning is obvious:
Content: A simple identifier can also be an expression. We use identifiers as variables to give names to values.
Content: When an identifier appears in an expression, this value is retrieved to provide a result for the expression.
Content: Here is an interaction with the Python interpreter that illustrates the use of variables as expressions.
100%|| 1/1 [00:00<00:00, 20.41it/s]
100%|| 1/1 [00:00<00:00, 1.06it/s]
Code: >>> x = 5
Code: >>> x
Code: 5
Code: >>> print x
Code: 5
Code: >>> print spam
100%|| 1/1 [00:00<00:00, 17.67it/s]
100%|| 1/1 [00:00<00:00, 4.81it/s]
Code: Traceback (innermost last):

100%|  | 1/1
[00:00<00:00, 58.55it/s]
100%|  | 1/1
[00:00<00:00, 4.89it/s]
Code: File "<pyshell#34>", line 1, in ?
Code: print spam
Code: NameError: spam
Code: >>>
Content: First the variable
Content: expression
Content: result when we put
Content: has not been assigned a value. Python cannot find a value, so it reports a
Content: no value with that name. A variable must always be assigned a value before it can be used in an expression.
Content: More complex and interesting expressions can be constructed by combining simpler expressions with
Content: operators
Content: The corresponding Python operators are:
Content: Here are some examples of complex expressions from
100%|  | 1/1
[00:00<00:00, 48.19it/s]
100%|  | 1/1
[00:00<00:00, 5.55it/s]
Code: 3.9 * x * (1 - x)
100%|  | 1/1
[00:00<00:00, 50.83it/s]
100%|  | 1/1
[00:00<00:00, 5.45it/s]
Code: 9.0 / 5.0 * celsius + 32
Content: Spaces are irrelevant within an expression. The last expression could have been written

Content: and the result would be exactly the same. Usually it’s a good idea to place some spaces in expressions to
Content: make them easier to read.
Content: Python’s mathematical operators obey the same rules of precedence and associativity that you learned
Content: in your math classes, including using parentheses to modify the order of evaluation. You should have lit-
Content: tle trouble constructing complex expressions in your own programs. Do keep in mind that only the round
Content: parentheses are allowed in expressions, but you can nest them if necessary to create expressions like this.
Content: If you are reading carefully, you may be curious why, in her temperature conversion program, Suzie
Content: Programmer chose to write
Content: different results. This mystery will be discussed in Chapter 3. If you can’t stand the wait, try them out for
Content: yourself and see if you can figure out what’s going on.
100%|| 1/1 [00:00<00:00, 63.66it/s]
100%|| 1/1 [00:00<00:00, 6.51it/s]
Heading: 2.4
Heading: Output Statements
Content: Now that you have the basic building blocks, identifier and expression, you are ready for a more complete
Content: description of various Python statements. You already know that you can display information on the screen
Content: using Python’s
Content: has a precise set of rules for the syntax (form) and semantics (meaning) of each statement. Computer scientists
Content: have developed sophisticated notations called
Content: book we will rely on a simple template notation to illustrate the syntax of statements.
Content: Here are the possible forms of the
Times,serif
9.96264
Content: 2.5. ASSIGNMENT STATEMENTS
Content: 17
Code: print
Code: print <expr>
Code: print <expr>, <expr>, ..., <expr>

Code: print <expr>, <expr>, ..., <expr>,
Content: In a nutshell, these templates show that a
100%|  | 1/1
[00:00<00:00, 49.13it/s]
100%|  | 1/1
[00:00<00:00, 5.20it/s]
Content: “slots” that are filled in by other fragments of Python code. The name inside the brackets indicate what is
Content: missing;
Content: optionally ended with a comma. That is all there is to know about the syntax of
Content: As far as semantics, a
Content: are evaluated left to right, and the resulting values are displayed on a single line of output in a left-to-right
Content: fashion. A single blank space character is placed between the displayed values.
Content: Normally, successive
Content: method, multiple
Content: Putting it all together, this sequence of
100%|  | 1/1
[00:00<00:00, 34.59it/s]
100%|  | 1/1
[00:00<00:00, 6.24it/s]
Code: print 3+4
100%|  | 1/1
[00:00<00:00, 46.14it/s]
100%|  | 1/1
[00:00<00:00, 5.31it/s]
Code: print 3, 4, 3 + 4
Code: print
Code: print 3, 4,
Code: print 3+4
100%| 


| 1/1 [00:00<00:00, 23.76it/s]
100%| 
| 1/1 [00:00<00:00, 5.96it/s]
Code: print "The answer is", 3 + 4
Content: produces this output
Code: 7
Code: 3 4 7
Code: 3 4 7
Code: The answer is 7
Content: That last
Content: requires a sequence of expressions. That means
Content: an expression, but it doesn’t produce a number. Instead, it produces another kind of data called a
Content: A sequence of characters enclosed in quotes is a string literal. Strings will be discussed in detail in a later
Content: chapter. For now, consider this a convenient way of labeling output.
Heading: 2.5
Heading: Assignment Statements
100%| 
| 1/1 [00:00<00:00, 20.86it/s]
100%| 
| 1/1 [00:00<00:00, 6.54it/s]
Subheading: 2.5.1 Associated Heading: Assignment Statements
Subheading: Simple Assignment Associated Heading: Assignment Statements
Content: One of the most important kinds of statements in Python is the assignment statement. We’ve already seen a
Content: number of these in our previous examples. The basic assignment statement has this form:
Code: <variable> = <expr>
Content: Here
Content: expression on the right side is evaluated to produce a value, which is then associated with the variable named
Content: on the left side.
Content: Here are some of the assignments we’ve already seen.
100%| 


| 1/1 [00:00<00:00, 48.53it/s]
100%| 
| 1/1 [00:00<00:00, 6.31it/s]
Code: x = 3.9 * x * (1 - x)
100%| 
| 1/1 [00:00<00:00, 34.71it/s]
100%| 
| 1/1 [00:00<00:00, 5.57it/s]
Code: fahrenheit = 9.0 / 5.0 * celsius + 32
Code: x = 5
Times,serif
9.96264
Content: 18
Content: CHAPTER 2. WRITING SIMPLE PROGRAMS
Content: A variable can be assigned many times. It always retains the value of the most recent assignment. Here
Content: is an interactive Python session that demonstrates the point:
Code: >>> myVar = 0
Code: >>> myVar
100%| 
| 1/1 [00:00<00:00, 43.33it/s]
100%| 
| 1/1 [00:00<00:00, 6.28it/s]
Code: 0
Code: >>> myVar = 7
Code: >>> myVar
Code: 7
100%| 
| 1/1 [00:00<00:00, 32.06it/s]
100%| 

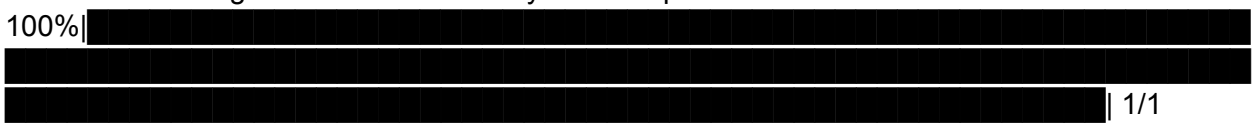

| 1/1 [00:00<00:00, 5.31it/s]
Code: >>> myVar = myVar + 1
Code: >>> myVar
Code: 8
Content: The last assignment statement shows how the current value of a variable can be used to update its value. In
Content: this case I simply added one to the previous value. The
Content: similar, though a bit more complex. Remember, the values of variables can change; that’s why they’re called
Content: variables.
Subheading: 2.5.2 Associated Heading: Assignment Statements
Subheading: Assigning Input Associated Heading: Assignment Statements
Content: The purpose of an input statement is to get some information from the user of a program and store it into a
Content: variable. Some programming languages have a special statement to do this. In Python, input is accomplished
Content: using an assignment statement combined with a special expression called
Content: standard form.
100%|

| 1/1 [00:00<00:00, 46.60it/s]
100%|

| 1/1 [00:00<00:00, 4.47it/s]
Code: <variable> = input(<prompt>)
Content: Here
Content: (i.e., some text inside of quotation marks).
Content: When Python encounters an
Content: prompt on the screen.
Content: Python then pauses and waits for the user to type an expression and press the
100%|

| 1/1 [00:00<00:00, 43.29it/s]
100%|

| 1/1 [00:00<00:00, 4.47it/s]
Code: &#xfffd;
100%|


 1/1
[00:00<00:00, 32.82it/s]
100%  1/1
[00:00<00:00, 6.87it/s]
Subheading: &#xfffd; Associated Heading: Assignment Statements
Content: key. The expression typed by the user is then evaluated to produce the result of the
Content: This sounds complicated, but most uses of
Content: statements are used to get numbers from the user.
100%  1/1
[00:00<00:00, 38.99it/s]
100%  1/1
[00:00<00:00, 6.55it/s]
Code: x = input("Please enter a number between 0 and 1: ")
100%  1/1
[00:00<00:00, 35.86it/s]
100%  1/1
[00:00<00:00, 4.74it/s]
Code: celsius = input("What is the Celsius temperature? ")
Content: If you are reading programs carefully, you probably noticed the blank space inside the quotes at the end
Content: of these prompts. I usually put a space at the end of a prompt so that the input that the user types does not
Content: start right next to the prompt. Putting a space in makes the interaction easier to read and understand.
Content: Although these two examples specifically prompt the user to enter a number, a number is just a numeric
Content: literal—a simple Python expression. In fact, any valid expression would be just as acceptable. Consider the
Content: following interaction with the Python interpreter.
100%  1/1
[00:00<00:00, 22.78it/s]

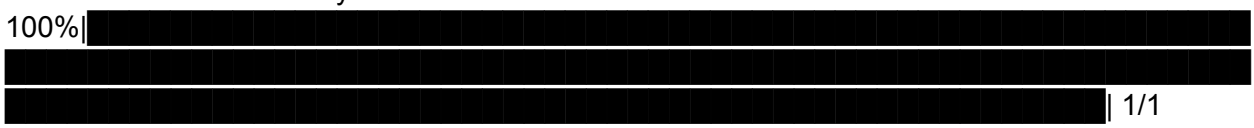
100%|[redacted]| 1/1
[00:00<00:00, 4.84it/s]
Code: >>> ans = input("Enter an expression: ")
100%|[redacted]| 1/1
[00:00<00:00, 35.24it/s]
100%|[redacted]| 1/1
[00:00<00:00, 4.99it/s]
Code: Enter an expression: 3 + 4 * 5
Code: >>> print ans
Code: 23
Code: >>>
Content: Here, when prompted to enter an expression, the user typed “3 + 4 * 5.” Python evaluated this expression and
Content: stored the value in the variable
Content: In a way, the
Content: result as if we had simply done
Content: at the time the statement was executed instead of being determined when the statement was written by the
Content: programmer. Thus, the user can supply formulas for a program to evaluate.
Times,serif
9.96264
Content: 2.5. ASSIGNMENT STATEMENTS
Content: 19
Subheading: 2.5.3 Associated Heading: Assignment Statements
Subheading: Simultaneous Assignment Associated Heading: Assignment Statements
Content: There is an alternative form of the assignment statement that allows us to calculate several values all at the
Content: same time. It looks like this:
Code: <var>, <var>, ..., <var> = <expr>, <expr>, ..., <expr>
Content: This is called
Content: the right-hand side and then assign these values to the corresponding variables named on the left-hand side.
Content: Here’s an example.
100%|[redacted]| 1/1
[00:00<00:00, 21.10it/s]

100%|  | 1/1
[00:00<00:00, 6.96it/s]
Code: sum, diff = x+y, x-y
Content: Here
Content: This form of assignment seems strange at first, but it can prove remarkably useful. Here’s an example.
Content: Suppose you have two variables
Content: currently stored in
Content: this could be done with two simple assignments.
Code: x = y
Code: y = x
Content: This doesn’t work. We can trace the execution of these statements step-by-step to see why.
Content: Suppose
Content: variables change. The following sequence uses comments to describe what happens to the variables as these
Content: two statements are executed.
Code: # variables
Code: x
100%|  | 1/1
[00:00<00:00, 31.21it/s]
100%|  | 1/1
[00:00<00:00, 5.39it/s]
Code: y
Code: # initial values
100%|  | 1/1
[00:00<00:00, 44.45it/s]
100%|  | 1/1
[00:00<00:00, 5.42it/s]
Code: 2
100%|  | 1/1
[00:00<00:00, 44.23it/s]

100%|  | 1/1 [00:00<00:00, 4.58it/s]
Code: 4
Code: x = y
Code: # now
Code: 4
Code: 4
Code: y = x
Code: # final
Code: 4
Code: 4
Content: See how the first statement clobbers the original value of
Content: assign
Content: One way to make the swap work is to introduce an additional variable that temporarily remembers the
Content: original value of
Code: temp = x
Code: x = y
Code: y = temp
Content: Let’s walk-through this sequence to see how it works.
Code: # variables
Code: x
Code: y
Code: temp
Code: # initial values
Code: 2
Code: 4
Code: no value yet
Code: temp = x
Code: #
Code: 2
Code: 4
Code: 2
Code: x = y
Code: #
Code: 4
Code: 4
Code: 2
Code: y = temp
Code: #
Code: 4
Code: 2

Code: 2
Content: As you can see from the final values of
Content: This sort of three-way shuffle is common in other programming languages. In Python, the simultaneous
Content: assignment statement offers an elegant alternative. Here is a simpler Python equivalent:
Code: x, y = y, x
Times,serif
9.96264
Content: 20
Content: CHAPTER 2. WRITING SIMPLE PROGRAMS
Content: Because the assignment is simultaneous, it avoids wiping out one of the original values.
Content: Simultaneous assignment can also be used to get multiple values from the user in a single
Content: sider this program for averaging exam scores:
Code: # avg2.py
Code: #
Code: A simple program to average two exam scores
Code: #
Code: Illustrates use of multiple input
100%| | 1/1 [00:00<00:00, 21.44it/s]
100%| | 1/1 [00:00<00:00, 4.78it/s]
Code: def main():
Code: print "This program computes the average of two exam scores."
100%| | 1/1 [00:00<00:00, 44.11it/s]
100%| | 1/1 [00:00<00:00, 5.32it/s]
Code: score1, score2 = input("Enter two scores separated by a comma: ")
100%| | 1/1 [00:00<00:00, 42.90it/s]

100%| | 1/1
[00:00<00:00, 5.91it/s]
Code: average = (score1 + score2) / 2.0
Code: print "The average of the scores is:", average
100%| | 1/1
[00:00<00:00, 43.15it/s]
100%| | 1/1
[00:00<00:00, 4.91it/s]
Code: main()
Content: The program prompts for two scores separated by a comma. Suppose the user types
Content: the
Code: score1, score2 = 86, 92
Content: We have gotten a value for each of the variables in one fell swoop. This example used just two values, but it
Content: could be generalized to any number of inputs.
Content: Of course, we could have just gotten the input from the user using separate input statements.
100%| | 1/1
[00:00<00:00, 22.04it/s]
100%| | 1/1
[00:00<00:00, 5.65it/s]
Code: score2 = input("Enter the second score: ")
Content: In some ways this may be better, as the separate prompts are more informative for the user. In this example
Content: the decision as to which approach to take is largely a matter of taste. Sometimes getting multiple values in a
Content: single
Heading: 2.6
Heading: Definite Loops
Content: You already know that programmers use loops to execute a sequence of statements several times in succession.
Content: The simplest kind of loop is called a
Content: That is, at the point in the program when the loop begins, Python knows how many times to go around

Content: executed exactly ten times.
100%|  | 1/1 [00:00<00:00, 40.73it/s]
100%|  | 1/1 [00:00<00:00, 7.91it/s]
Code: for i in range(10):
Code: x = 3.9 * x * (1 - x)
Code: print x
Content: This particular loop pattern is called a
Content: considering this example in detail, let’s take a look at what
Content: A Python
Code: for <var> in <sequence>:
Code: <body>
Content: The body of the loop can be any sequence of Python statements. The start and end of the body is indicated
Content: The meaning of the
Content: once you get the hang of it.
Content: The variable after the keyword
(venv) (base) harsh@harsh-Inspiron-7570:~/Skillevant_VirtualEnv/Codebase$ pip freeze > requirements.txt