



Computer Programming in Python

Chapter 3

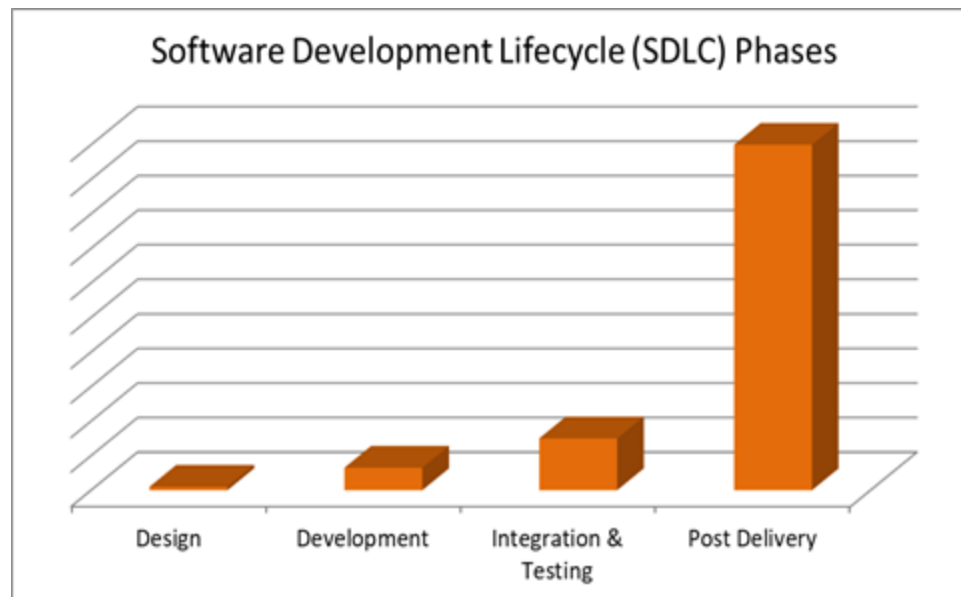
Getting Started in Python

Chapter 3 Getting Started in Python

- Design
 - A few minutes of design time will save hours of programming (testing and debugging) because:
 - A solution is in mind before any code has been written
 - The solution is viewed as a comprehensive program
 - Often one part of a program has a direct impact on other parts of the program
 - A poor design in one area can impact other areas, introduce bugs, require hours of debugging, and force re-writing of code

Chapter 3 Getting Started in Python

- Design
 - Bugs (issues) discovered during the design phase are easier to find and faster to eliminate (cost increases)



Chapter 3 Getting Started in Python

- Design/Development Tools - Commenting Code
 - Another tool for development in addition to pseudocode and flowcharts, is placing comments within the code
 - They are also often required
 - **Comments**
 - Lines of code that are not executed
 - Provided for human readers
 - Used to clarify values being used or sections of the code
 - Used to explain complex operations

Chapter 3 Getting Started in Python

- **Commenting Code is Important**
 - Most software is maintained, updated, and expanded
 - Code is written once, but is read many times
 - The person who wrote the code may not be the person making the modifications
 - The person who wrote the code may not remember why a section was written a certain way or why a specific value was used
 - Adding comments to code while it is being written can save hours of reading through the lines later when the code is being changed

Code is written once, but read many times

Chapter 3 Getting Started in Python

- **Commenting Code**

- Comments can also be used as a development tool
- Pseudocode can be written in the program as a comment
 - Act as a place-holder or reminder
 - Later replaced by actual code

```
File Edit Format Run Options Window Help
1
2
3 # Implement the prompt to get input
4
5 # Compute the value
6
7 # Display the result
8
9
```

Chapter 3 Getting Started in Python

- **Comments vs Executable Code**
 - Source code written in programs is often referred as SLOCs, and includes comments as well as executable lines
 - The executable lines by themselves are referred to as ELOCs

SLOC – Source Line of Code (include all text)

ELOC – Executable Line of Code (omits comments)

Chapter 3 Getting Started in Python

- Comments in Python
 - Can be single-line or multi-line
 - Single-line and end-of-line comments begin with the pound sign (octothorp)

```
# This is a single line comment in Python  
print("Hello")    # This is an end of line comment
```


Chapter 3 Getting Started in Python

- Comments in Python
 - Multiline comments are surrounded by three single or three double quotes

```
# This is a multiline comment in Python  
# using the pound sign
```

```
"""  
This is a multiline comment in Python  
using three sets of double quotes  
"""
```

Chapter 3 Getting Started in Python

- Comments in Python
 - Are ignored by the interpreter
 - IDEs will color code comments to highlight them

```
# Simulates draining the canal where level is
# the water level in feet (float) and opening is
# the flood gate opening in feet (integer).
# drainage occurs at 0.03 feet per minute times
# the gate opening in feet
def simulate(level, opening):
    drainage = 0.03 * opening
    minutes = 1

    while level > 3.3:|
```

Chapter 3 Getting Started in Python

- Displaying Output
 - The *print* function in Python displays output to the shell
 - A *function* is code that exists and is *called* in order to execute

```
>>> print('The Python Language')  
The Python Language  
>>>
```

To execute a Function, it must be called

Chapter 3 Getting Started in Python

- Displaying Output

```
>>> print('The Python Language')  
The Python Language  
>>>
```

- The text inside the parentheses and quotes is a *string* (group of characters) which is an *argument* passed to the print function
 - Any piece of data passed to a function is referred to as an argument

Chapter 3 Getting Started in Python

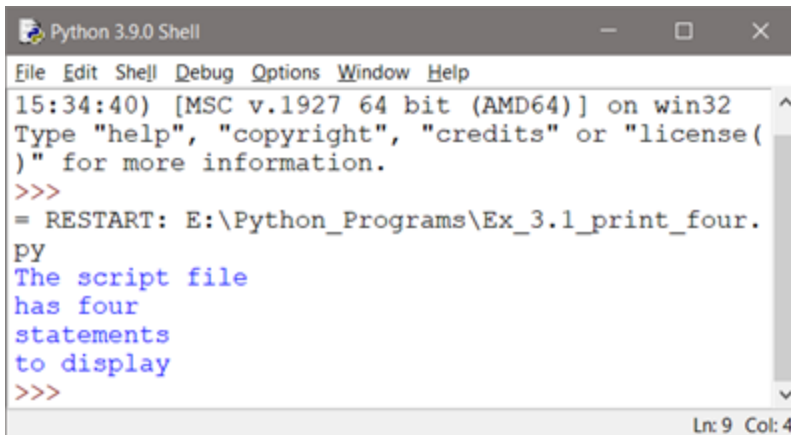
- Displaying Output

```
>>> print('The Python Language')  
The Python Language  
>>>
```

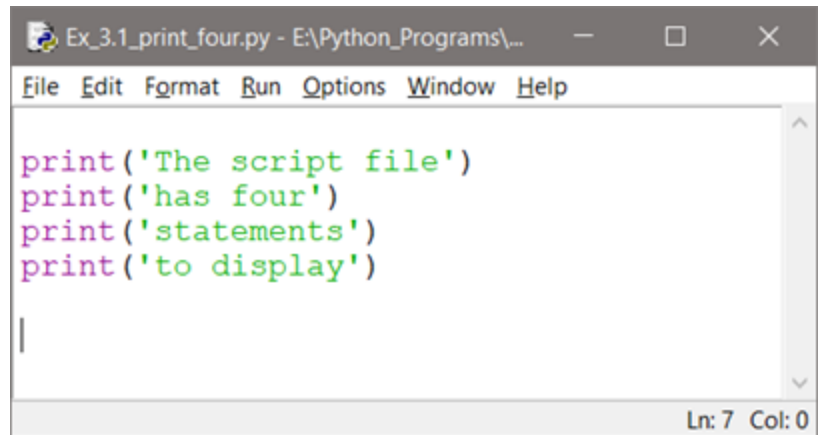
- The argument contains the item or items to display (print)
 - In this case “The Python Language”
- Note: the print function will automatically add a line feed in the output

Chapter 3 Getting Started in Python

- Displaying Output Using Files
 - To use the interpreter in *script mode*, a new file is created using **File -> New** from the shell menu
 - Below, the print function is called four different times and is passed four different strings of characters



```
Python 3.9.0 Shell
File Edit Shell Debug Options Window Help
15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license(
)" for more information.
>>>
= RESTART: E:\Python_Programs\Ex_3.1_print_four.
PY
The script file
has four
statements
to display
>>>
Ln: 9 Col: 4
```



```
Ex_3.1_print_four.py - E:\Python_Programs\...
File Edit Format Run Options Window Help

print('The script file')
print('has four')
print('statements')
print('to display')

Ln: 7 Col: 0
```

Chapter 3 Getting Started in Python

- Strings
 - A *string* is a sequence of characters and is the *str* data type in Python
 - When a string is written in the code surrounded by quotes, it is referred to as a *string literal*
 - Single quotes or double quotes can surround strings
 - The PEP 8 Style Guide for Python does not make a recommendation for which to use, except to be consistent
 - There are cases when one choice is required (next slide)



Chapter 3 Getting Started in Python

- **Displaying Quotes**

- For a single quote (apostrophe) in the output, the string is surrounded with double quotes
- For double quotes in output, surround the string with single quotes
- To display both, use three sets of quotes

```
print(" Double quotes for an apostrophe like the word can't ")  
print(' Single quotes for double quotes like "Quotes" ')  
print('"' I'm displaying "both" "')
```

```
Double quotes for an apostrophe like the word can't  
Single quotes for double quotes like "Quotes"  
I'm displaying "both"  
>>> |
```


Chapter 3 Getting Started in Python

- Variables

- In the earlier example to display errors, a name was entered into the program and was stored in a *variable* called *name* (code repeated here)

```
name = input('Please enter your name: ')
```

- The input function extracted what was entered on the keyboard when *Enter* was pressed, and *assigned* it to *name*

Chapter 3 Getting Started in Python

- Variables

```
name = input('Please enter your name: ')
```

- This is called an assignment statement
- The equal sign is the ***assignment operator***
- The right side of an assignment statement is evaluated first, and the result is assigned to the left side

Chapter 3 Getting Started in Python

- Variables
 - Variables are used to allocate memory and store information that the program will use
 - What they store can vary as the program runs
 - A variable name is a name given by the programmer to a piece of data that is stored in memory

```
my_string = 'something'
```



Variable
Name

Chapter 3 Getting Started in Python

- Variables

- The assignment operator is used to assign a value to a variable
- Below, the word “something” is assigned to a variable declared with the name “my_string”

```
my_string = 'something'
```

Define a variable by assigning a value

Chapter 3 Getting Started in Python

- Variables

- Below, the value 15 is assigned to a variable declared with the name “number”

- This *defines* the variable
 - The value 15 will be stored by the computer and it will be accessed (used by the program) using the name ‘number’
 - The second line prints the value stored in the variable ‘number’

```
number = 15
```

```
print(number)
```

Chapter 3 Getting Started in Python

- Variables

```
number = 15
```

```
print(number)
```

- Notice that there are no quotes around the word number in the print statement
- The variable number is passed to the print function and the value that was stored in memory will be displayed in the shell

```
15  
>>>
```

Chapter 3 Getting Started in Python

- Variables
 - Programs often use multiple variables to store multiple values and each variable must have a distinct name
 - The PEP 8 Style Guide lists multiple styles for naming variables in Python, but none is recommended or preferred
 - Programmers prefer using all lowercase letters with underscores separating words

Chapter 3 Getting Started in Python

- Variables

- The preferred style for Python is all lower case letters with multi-word names separated by an underscore
- Be consistent

perimeter tax_rate user_age

discount_price radius hourly_rate

gross_pay tickets_sold diameter

Variable naming using lower case letters and underscores

Chapter 3 Getting Started in Python

- Variables

- The following (typical) style will be used:

- If the variable name is one word, it will be all lowercase
- If two or more words are used, they will be separated by an underscore

```
number = 15  
print(number)
```

```
first_variable = 20  
print(first_variable)
```

```
second_variable = 30  
print(second_variable)
```

Chapter 3 Getting Started in Python

- Variables – Python is Case Sensitive

- Case errors and misspellings will be caught by the interpreter and a Traceback error will identify the line number for the error
- As an example, number is defined with all lowercase letters, but an uppercase letter is used in the print statement

```
number = 15  
print(Number)    # not the same as number
```

- The error shows that 'Number' with an uppercase 'N' is not defined

```
NameError: name 'Number' is not defined  
>>>
```

Chapter 3 Getting Started in Python

- Variables
 - A similar error occurs when a variable has not been assigned a value
 - Python does not know what type of data to store without an assignment statement
 - It determines how to store the data for a variable in memory based on the type of data it is assigned

Chapter 3 Getting Started in Python

- Variable Data Types
 - In addition to the string data type (and others covered later), the Python numeric data types include:
 - int (integer)
 - Whole numbers
 - 4 Bytes of memory allocated
 - float (floating point number)
 - Numbers with a fractional part (decimal)
 - 8 Bytes of memory allocated

Chapter 3 Getting Started in Python

- Variables

- In the code below, the variable is declared without assigning it a value (defining it), and it is then passed to the print function

```
first_variable # not defined with an assignment  
  
print(first_variable)
```

- Since a value was not assigned to the variable, it is undefined to the interpreter.

```
NameError: name 'first_variable' is not defined  
>>>
```

Chapter 3 Getting Started in Python

- Variables

- Three variables declared with different data types

```
var1 = 15                # an integer
print(var1)

var2 = 12.3              # a floating point number
print(var2)

var3 = 'some text'      # a string
print(var3)

15
12.3
some text
>>>
```

Chapter 3 Getting Started in Python

- Variables

- In Python, a variable can be assigned one data type and then another

```
number = 15                # assign 15 to number
print(number)

number = 'Now a string'   # assign a string
print(number)

number = 2                # assign a number again
number = number + 2       # add 2 to number
print(number)
```

```
15
Now a string
4
>>>
```

Chapter 3 Getting Started in Python

- Data Types Categorize values in memory

Type	Description	Example
int	integers (whole numbers)	3, 18, 56, 1234
float	real numbers (floating point)	7.56, 12.345
str	sequences of characters (strings)	“Python”, “two words”
bool	logical values (Boolean)	True, False

Chapter 3 Getting Started in Python

- Data Types
 - Floating point numbers are stored with double precision although the data type *double* is not used in Python
 - The *bool* data type can be assigned either True or False

```
valid_data = False
```

```
found_num = True
```

Chapter 3 Getting Started in Python

- Variable Name Rules
 - None of Python's key words can be used
 - There cannot be any spaces in the name
 - The first character must be a letter (or an underscore)
 - Uppercase and lowercase letters are distinct
 - Descriptive names are preferred
 - Longer is usually better

A variable name should describe what it stores

Chapter 3 Getting Started in Python


- Variable Names

Name		Comment
x		does not describe the data being stored
6pack		cannot begin with a digit
sixPack	✓	proper, but not preferred
my_num	✓	proper, but ambiguous
how?many		can only contain letters, digits, and underscores
temperature	✓	proper naming
gross_pay	✓	proper naming

Chapter 3 Getting Started in Python

- Named Constants
 - A value in programming that is not changed by the program
 - Defined using all uppercase letters with underscores between words
 - Used to eliminate magic numbers
 - A *magic number* is a literal number in a program without an obvious meaning

circumference = 2 * PI * 3963.2 ?



Chapter 3 Getting Started in Python

- Magic Numbers
 - When a program is being modified and an expression uses a literal value, it may be difficult to determine what the number means even by the original programmer
 - The meaning of 3963.2 below is unknown
 - Since the equation results in a circumference, it appears to be a radius, but...

$$\text{circumference} = 2 * \text{PI} * 3963.2$$

Chapter 3 Getting Started in Python

- Named Constants
 - With a named constant in the code, the meaning is clear
 - It is defined and then used in place of the literal number wherever needed in the program

```
EARTH_RADIUS = 3963.2
```

```
circumference = 2 * PI * EARTH_RADIUS
```

Named constants remove ambiguity (magic numbers)

Chapter 3 Getting Started in Python

- Named Constants
 - Named constants are also used to ensure that the same value is used throughout the program and by all programmers
 - Example: When multiple programmers are working on a program that requires the radius of the earth in various equations, they can use the named constant instead of typing in different values
 - The earth is not a sphere and there are multiple values for its radius

Chapter 3 Getting Started in Python

- Named Constants – more benefits
 - Named constants also prevent typographical errors when the same value is being used multiple times
 - The greater the number of times it is typed, the greater the chances of mistyping
 - When a new value is needed for the constant, the change is made in a single place in the code

Chapter 3 Getting Started in Python

- Named Constants – more benefits

Example: the scientist overseeing the program using `EARTH_RADIUS` decides that the equatorial radius being used in the program should be changed to the pole radius of 3950.0.

- The value will only need to be changed to the new value in one place in the code
- This eliminates the possibility of typographical errors when changing all of the equations that use the value, time spent debugging the program if an equation is overlooked and the output is incorrect.

Chapter 3 Getting Started in Python

Displaying Output and the `print()` Function

Chapter 3 Getting Started in Python

- Printing Multiple Elements

- To pass multiple arguments to the print function, the plus sign or comma are used depending on the data types
- A comma is used when a literal string and numeric value are being displayed

```
num = 1000
print('The number is', num)

print('The number is', 123)
```

```
The number is 1000
The number is 123
>>>
```

Chapter 3 Getting Started in Python

- Printing Multiple Elements

- In the print statements, there is no space after the word “is”, but it is included in the output

```
num = 1000
print('The number is', num)

print('The number is', 123)
```

```
The number is 1000
The number is 123
>>>
```

- The print function automatically adds a space which is the default separator (when none is provided) for multiple arguments

Chapter 3 Getting Started in Python

- Printing Multiple Elements
 - Below, an integer variable is surrounded by two string literals
 - Commas separate the arguments, and they are automatically separated by spaces in the output

```
tics = 123
print('Today we sold', tics, 'tickets')
```

```
Today we sold 123 tickets
>>>
```

Chapter 3 Getting Started in Python

- Separators
 - When the space is not needed or another separator is required, the argument *sep=* can be passed to the print function
 - The separator assigned will be used between all of the arguments

```
print('Today we sold', tics, 'tickets', sep='')
```



Chapter 3 Getting Started in Python

- Separators

- The first print statement below uses *sep=""* to tell the print function to separate elements using *nothing* (there is no space between the two single quotes)
- In the second print statement, the print function is told to use the semicolon as the separator

```
tics = 123
print('Today we sold', tics, 'tickets', sep='')
print('Today we sold', tics, 'tickets', sep=';')
```



Chapter 3 Getting Started in Python

- Separators

- Note the output for the second line

```
tics = 123
print('Today we sold', tics, 'tickets', sep='')
print('Today we sold', tics, 'tickets', sep=';')
```

```
Today we sold123tickets
Today we sold;123;tickets
>>>
```

- Any character or string can be used to separate the elements

Chapter 3 Getting Started in Python

- String Concatenation
 - In Python, the plus operator performs *concatenation*
 - Joins two or more strings together to form a single item
 - In the output below, note the missing spaces between the words
 - The strings were simply joined together

```
print('First' + 'second' + 'third')
```

```
Firstsecondthird
```

```
>>>
```

Chapter 3 Getting Started in Python

- String Facts
 - As a technical note, strings in Python are *immutable*
 - They cannot be changed
 - When a string is changed (as in concatenation), Python creates a new string and then points the variable name to the new string in memory
 - This is typically not an issue, and the interpreter will free the previously used memory

Chapter 3 Getting Started in Python

- String Facts

- As an example, Python creates a new string containing the combined words shown below and then points the variable name *'word1'* to the new string in memory

```
word1 = 'first'  
word1 = word1 + 'second'  
  
print(word1)
```

Chapter 3 Getting Started in Python

Formatting Output and Displaying Decimal Places

Chapter 3 Getting Started in Python

- Formatting Output
 - For dollar amounts or other numbers requiring decimal places are part of the output, the *format* function provides for output formatting
 - Two arguments are passed to the format function:
 - The numeric value or variable to be formatted
 - The format specification in quotes

```
print(format(5, '.2f'))
```

Chapter 3 Getting Started in Python

- Formatting Output
 - The format function returns a string holding the formatted number which can then be stored as a string in a variable or passed to the print function directly

```
text1 = format(1.2345, '.1f')  
  
print(format(5, '.2f'))
```

Chapter 3 Getting Started in Python

- Formatting Output
 - The “.2” in the specifier calls for the decimal and two decimal places to be used in the output, and “f” indicates formatting a floating-point number

```
print(format(5, '.2f'))  
  
num = 1.2345  
  
print(format(num, '.2f'))
```

5.00
1.23
>>>

Chapter 3 Getting Started in Python

- The Format Function Rounds
 - When required to fit within the number of decimal places specified for formatting, the number will be rounded (down from 5 and up from 6)

```
num = 123.98765

print(format(num, '.4f'))    123.9877
print(format(num, '.3f'))    123.988
print(format(num, '.2f'))    123.99
print(format(num, '.1f'))    124.0
>>>
```


Chapter 3 Getting Started in Python

- Formatting Output

- Values can be formatted when assigning them to variables
- The formatted variables can then be passed to the print function

```
text1 = format(1.2345, '.1f')
text2 = format(9.876, '.1f')
print(text1, text2)
```

```
1.2 9.9
9.88
>>>
```

```
num = 9.876
print(format(num, '.2f'))
```


Chapter 3 Getting Started in Python

- Formatting Output

- When the number being output requires a comma or commas, the specifier (a comma) is added prior to the decimal in the format specification

```
num = 12345678.99
print(format(num, ',.2f'))
```

12,345,678.99
>>>



Chapter 3 Getting Started in Python

- Formatting Output
 - To format integers, a “d” replaces the “f” in the specifier

```
print(format(1234, ',d'))
```

```
1,234  
>>>
```



Chapter 3 Getting Started in Python

- Formatting Output
 - For scientific notation, “e” is the specifier
 - Preceding it with a decimal point and an integer designates the number of places after the decimal to use in the output

```
num = 12345678.99
print(format(num, "e"))           1.234568e+07
print(format(num, ".1e"))        1.2e+07
>>>
```

Chapter 3 Getting Started in Python

- Formatting Output

- Numbers can also be specified as percentages using “%” and the number will be multiplied by 100 and include the “% sign in the format

- In the example, zero is used for the number of decimal places, but it can be any digit

```
stat = 0.27
print(format(stat, '.0%'))
```

27%
>>>


Chapter 3 Getting Started in Python

- Formatting Output

- For columns or right-aligned output, a minimum width specifier to use is placed before the decimal point
- Below, the width is designated as nine characters wide
 - A number with more than nine digits would not be trimmed

```
first = 1234.56
second = 654.32
```

```
print(format(first, "9.2f"))      1234.56
print(format(second, "9.2f"))    654.32
>>>
```



Chapter 3 Getting Started in Python

- Formatting Output
 - To suppress the automatic line feed added by print
 - Pass *end=* “ to the function

```
print('No lines', end='')    # suppress the line feed
print(' between', end='')   # suppress the line feed
print(' these.')
```

```
No lines between these.
>>>
```

Chapter 3 Getting Started in Python

- Formatting Output
 - To add necessary spaces

```
print('Do we ' + 'want ' + 'spaces?')  
print('Do we' + 'want' + 'spaces?')
```

```
Do we want spaces?  
Do wewantspaces?  
>>>
```


Chapter 3 Getting Started in Python

- **Escape Characters**
 - Special commands that begin with a backslash and can be embedded in strings for output

Escape	Result
<code>\n</code>	output a line feed
<code>\t</code>	output a tab
<code>\'</code>	output a single quote
<code>\"</code>	output a double quote
<code>\\</code>	output a backslash

Chapter 3 Getting Started in Python

- Escape Characters
 - The computer treats them as a single character
 - They must be inside quotes
 - Can add a tab, line feed, apostrophe, quotes, or backslash.

```
print("tab" + "\t" + "tab")           tab      tab
print("an extra line feed \n ")      an extra line feed
print("single quote \' ")           single quote '
print('double quote \" ')          double quote "
print("backslash \\")               backslash \
>>>
```

Often referred to as Escape Sequences

Chapter 3 Getting Started in Python

- Keyboard Input
 - In Python, *input* reads input from the keyboard
 - The function returns the value entered as a string which can then be converted if needed
 - The conversion is known as casting

```
words = input("Enter a string: ")
integer = int(input("Enter an integer: "))
decimal = float(input("Enter a float: "))
```

Chapter 3 Getting Started in Python

- Keyboard Input

```
words = input("Enter a string: ")
integer = int(input("Enter an integer: "))
decimal = float(input("Enter a float: "))
```

```
print() ← produces a line feed
print(words, integer, decimal)
print(integer + decimal) ← performs addition of
                           the integer and float
```

```
Enter a string: Hello
Enter an integer: 20
Enter a float: 1.78
```

```
Hello 20 1.78
21.78
>>>
```

Chapter 3 Getting Started in Python

- Type Conversion
 - Converting an item to another data type is referred to as *casting*
 - The *item* to be cast is surrounded with parentheses, and preceded by the data type needed

```
integer = int(input("Enter an integer: "))  
decimal = float(input("Enter a float: "))
```

Chapter 3 Getting Started in Python

- Type Conversion - Casting
 - Note that casting will only succeed if the *item* being cast is valid for the conversion
 - Trying to convert “Hello” to an integer will fail

```
item = 'word'  
print(int(item))
```

```
print(int(item))  
ValueError: invalid literal for int() with base 10: 'word'
```

Chapter 3 Getting Started in Python

- Type Conversion

- Casting examples:

```
first = 123.45
first_as_int = int(first)           # cast to an integer
print(first_as_int)
```

```
second = 5
second_as_float = float(second)    # cast to a float
print(second_as_float)
```

```
123
5.0
>>>
```

Chapter 3 Getting Started in Python

Programming Mathematical Equations

Chapter 3 Getting Started in Python

- Arithmetic Operators
 - Addition, subtraction, multiplication, two types of division, the modulus operator (modulo divide), and exponentiation

Operator	Description	Example	Result
+	addition	2 + 3	5
-	subtraction	72 - 12	60
*	multiplication	4 * 6	24
/	division	7 / 2	3.5
//	integer division	7 // 2	3
%	remainder (mod)	17 % 4	1
**	exponentiation	2 ** 3	8

Chapter 3 Getting Started in Python

- Arithmetic Operators
 - The values on the left and right side of the operator are referred to as *operands*
 - *Precedence* for operators in Python follows PEMDAS
 - Parenthetical expressions, followed by exponentiation, then multiplication, division, modulo division, and then addition and subtraction
 - Operators with the same precedence are handled left to right

Chapter 3 Getting Started in Python

- Arithmetic Operators

- Parenthesis can be used to force precedence or to add clarity to an equation

$4 * 2 + 15 / 5 - 2$

the result is 9

$4 * (2 + 15) / 5 - 2$

the result is 11.6

$4 * (2 + 15 / 5 - 2)$

the result is 12

$(4 * 2 + 15) / 5 - 2$

the result is 2.6

$4 * 2 + 15 / (5 - 2)$

the result is 13

$4 * (2 + 15) / (5 - 2)$

the result is 22.666

$(4 * 2) + (15 / 5) - 2$

the result is 9

Chapter 3 Getting Started in Python

- Arithmetic Operators
 - Including parentheses even when they align with precedence improves readability
 - Complex mathematical expressions can be broken into multiple statements to simplify the expression

Simplify complex expressions for readability

Chapter 3 Getting Started in Python

- Mixed-type Expressions
 - Results depend upon the data types in use

Two int values

the result is an int

Two float values

the result is a float

An int and float

the int is temporarily converted to a float and the result of the operation is a float.

Chapter 3 Getting Started in Python

- Mixed-type Expressions
 - Results depend upon the data types in use

```
number = 15                # stored as an integer
print(number)
```

```
number = number + 1.5     # float added to int
print(number)
```

```
15
16.5
>>>
```

Chapter 3 Getting Started in Python

- The Two Division Operators
 - A single forward slash is used for floating point division
 - Two forward slashes for integer division

Expression	Value of var1	Comment
<code>var1 = 10 // 5</code>	2	an integer
<code>var1 = 10 / 5</code>	2.0	a float
<code>var1 = 5.5 / 5</code>	1.1	a float
<code>var1 = 2.5 // 5</code>	0.0	fractional part discarded
<code>var1 = 5.5 // 5</code>	1.0	fractional part discarded
<code>var1 = -5.5 // 5</code>	-2.0	rounded away from zero

Chapter 3 Getting Started in Python

- The Two Division Operators
 - Override the mixed-type results
 - Dividing floating point numbers using the integer division operator will produce a floating point result, but it is truncated (not rounded)

Chapter 3 Getting Started in Python

- Rounding Numbers

- Python has a *round* function that will round numbers to an integer or to a specified number of places
- The number of decimal places is passed as the second argument to the function

```
number = round(9.4)
```

```
number is 9
```

```
number = round(9.6)
```

```
number is 10
```

```
number = round(12.2733, 2)
```

```
number is 12.27
```

```
number = round(12.2755, 2)
```

```
number is 12.28
```

```
number = round(-2.7777, 2)
```

```
number is -2.78
```

Chapter 3 Getting Started in Python

- Determining the Remainder
 - The *modulus* or remainder operator produces the remainder after division
 - Sometimes referred to as modulo divide
 - The operand on the left of the operator is divided by the operand on the right and the result is the remainder

`result = 5 % 2`

`result is 1`

Chapter 3 Getting Started in Python

- Determining the Remainder

- Remainder division is often used to determine if a number is even or odd, and to extract a digit from a number
- Python supports remainder division with floating point and negative numbers as well

result = 5 % 2

result is 1

result = 7 % 2.0

result is 1.0

result = -5 % 2

result is 1

result = -5 % -2

result is -1

result = 8.5 % 2

result is 0.5

Chapter 3 Getting Started in Python

- Exponents

- For raising a number to a power, the operator is two asterisks
- The operand to the left of the operator is raised to the power of the operand on the right

```
result = 2 ** 4
```

```
result is 16
```

```
result = 2 ** 4.0
```

```
result is 16.0
```

```
result = 2 ** 1.5
```

```
result is 2.8284271247...
```

```
result = 2 ** 0
```

```
result is 1
```

```
result = 2 ** -1
```

```
result is 0.5
```

Chapter 3 Getting Started in Python

- Algebraic Expressions
 - Converting mathematical expressions into Python code, the translation may require adding operators and parentheses to ensure the correct result
 - The expression $3xy$ in algebra would not be accepted by the Python interpreter
 - It would produce a syntax error
 - The multiplication operator must be inserted

$3xy$ translates to **$3 * x * y$**

Chapter 3 Getting Started in Python

- Algebraic Expressions
 - With fractions, precedence requires careful consideration to ensure that operations occur in the correct order

$$5xy$$

$$5 * x * y$$

$$x = \frac{3y}{2a}$$

$$x = 3 * y / 2 * a$$

$$x = \frac{y + 5}{z - 3}$$

$$x = (y + 5) / (z - 3)$$

$$x = \frac{n(n - 1)y^2}{2 + n}$$

$$x = (n * (n - 1) * y ** 2) / (2 + n)$$

Chapter 3 Getting Started in Python

- Algebraic Expressions
 - With extremely complex equations, breaking the expression into parts may be the best course of action

$$x = \frac{3y(z + 4)}{2a(zy)}$$

```
top = 3 * y * (z + 4)
bottom = 2 * a * z * y
result = top / bottom
```

Chapter 3 Getting Started in Python

- Breaking Long Statements or Expressions
 - Use the line continuation indicator (backslash)
 - Not necessary when a statement is enclosed in parenthesis as in the second and third examples.

```
wind_chill = 35.74 + (0.6215 * tempF) - (35.75 * (wind_speed**0.16)) + \
            0.4275 * tempF * (wind_speed**0.16)
```

```
result = (item1 + item2 + item3 +
          item4 + item5)
```


Chapter 3 Getting Started

Chapter 3 Getting Started in Python