PYTHON PROGRAMMING NOTES

SYLLABUS:

MODULE 1

Introduction to Python - The IDLE Python Development Environment - The Python Standard Library - Literals - Numeric Literals - String Literals - Control Characters - String Formatting - Implicit and Explicit Line Joining Variables and Identifiers - Variable Assignment and Keyboard Input- Identifier-Keywords and Other Predefined Identifiers in Python – Operators - Various Operators - Relational Operators-Membership Operators – Boolean Operators - Expression and Data Types -Operator Precedence and Boolean Expressions - Operator Associativity - Mixed-Type Expression

MODULE 2

Control Structure -Selection Control- If Statement - Indentation in Python - Multi-Way Selection - Iterative Control - While Statement - Input Error Checking - Infinite loops - Definite vs. Indefinite Loops

MODULE 3

List Structures - Common List Operations - List Traversal - Lists (Sequences) in Python- Python List Type - Tuples- Sequences- Nested Lists Iterating Over Lists (Sequences) in Python - For Loops - The Built-in range Function - Iterating Over List Elements vs. List Index Values-While Loops and Lists (Sequences) - Dictionaries and sets

MODULE 4

Defining Functions - Calling Value-Returning Functions - Calling Non-Value-Returning Functions - Parameter Passing - Keyword Arguments in Python - Default Arguments in Python - Variable Scope - Recursive functions - Exception Handling -The Propagation of Raised Exceptions - Catching and Handling Exceptions -Exception Handling and User Input

MODULE 5

String Processing - String Traversal - String-Applicable Sequence Operations -String Methods - Using Text Files - Opening Text Files - Reading Text Files - Writing Text Files

TEXT BOOK

1, Charles Dierbach, Introduction to Computer Science using Python , Wiley First Edition (2015), ISBN-10: 81265560132015

REFERENCE BOOKS

1, Zed A.Shaw, Learn Python the Hard Way Paperback, Pearson Education, Third Edition edition (2017), ISBN-10: 9332582106

2. Paul Barry, Head First Python, O' Reilly Publishers, First Edition, 2010, ISBN:1449382673.

MODULE 1

Introduction to Python - The IDLE Python Development Environment - The Python Standard Library - Literals - Numeric Literals - String Literals - Control Characters - String Formatting - Implicit and Explicit Line Joining Variables and Identifiers - Variable Assignment and Keyboard Input- Identifier-Keywords and Other Predefined Identifiers in Python – Operators - Various Operators - Relational Operators-Membership Operators – Boolean Operators - Expression and Data Types -Operator Precedence and Boolean Expressions - Operator Associativity - Mixed-Type Expression

INTRODUCTION TO PYTHON

- Guido van Rossum (Figure 1-26) is the creator of the Python programming language, first released in the early 1990s.

FIGURE 1-26 Guido van Rossum

- The development environment IDLE provided with Python (discussed below) comes from the name of a member of the comic group.
- Python has a simple syntax.
- Python programs are clear and easy to read.
- Python provides powerful programming features, and is widely used.
- Companies and organizations that use Python include YouTube, Google, Yahoo, and NASA. Python is well supported and freely available at www.python.org.

THE IDLE PYTHON DEVELOPMENT ENVIRONMENT:

- IDLE is an integrated development environment (IDE) .
- An IDE is a bundled set of software tools for program development.
- This typically includes ***an editor for creating and modifying programs***, ***a translator f or executing programs***, and **a program debugger**.

- A debugger provides a means of taking control of the execution of a program to aid in finding program errors.
- Python is most commonly translated by use of an interpreter.
- Thus, Python provides the very useful ability to execute in interactive mode.
- The window that provides this interaction is referred to as **the Python shell**.

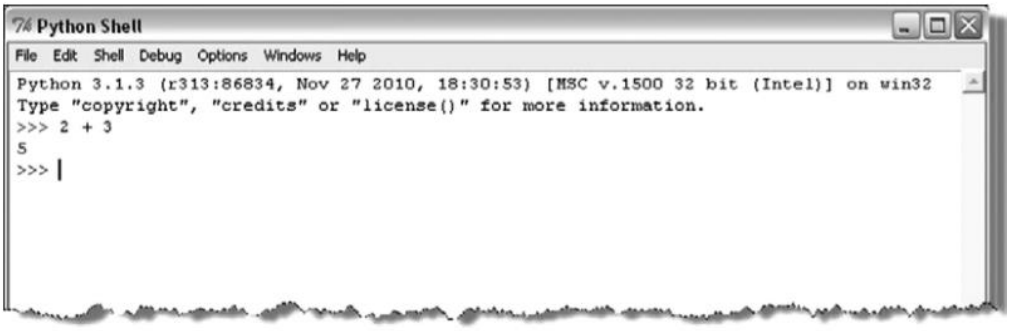

FIGURE 1-27 Python Shell

- Here, the expression 2 + 3 is entered at the shell prompt (...), which immediately responds with the result 5.

THE PYTHON STANDARD LIBRARY:

- ***The Python Standard Library is a collection of modules, each providing specific functionality beyond what is included in the core part of Python.***
- For example, the **math module** provides additional **mathematical functions**.
- The **random module** provides the ability to generate **random numbers**, useful in programming.
- In order to utilize the capabilities of a given module in a specific program, an import statement is used as shown in Figure 1-28.
- The example in the figure shows the use of the import math statement to gain access to a particular function in the math module, the factorial function. The syntax for using the factorial function is **math.factorial(n),** for some positive integer **n**.

A BIT OF PYTHON:

We introduce a bit of Python, just enough to begin writing some simple programs.

Since all computer programs input data, process the data, and output results, we look at the notion of a variable, how to perform some simple arithmetic calculations, and how to do simple input and output.

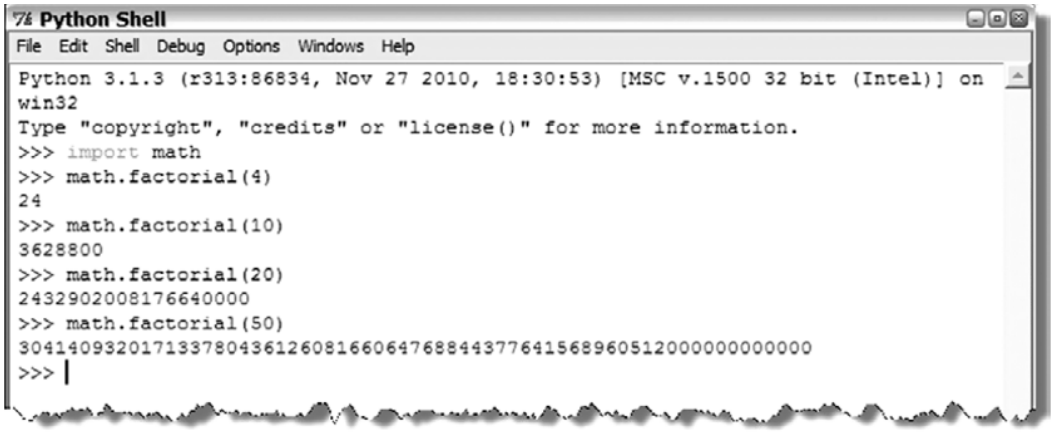

FIGURE 1-28 Using an import statement

VARIABLES:

- One of the most fundamental concepts in programming is that of a variable.
- A simple description of a variable is "a name that is assigned to a value," as shown below,
- **n = 5** variable n is assigned **the value 5**

Some Basic Arithmetic Operators:

- The common arithmetic operators in Python are + (addition), - (subtraction), * (multiplication), / (division), and ** (exponentiation).
- Addition, subtraction, and division use the same symbols as standard mathematical notation,
- 10 + 20 25 - 15 20 / 10
- There is also the symbol // for truncated division.
- For multiplication and exponentiation, the asterisk (*) is used.
- 5 * 10 (5 times 10) 2 ** 4 (2 to the 4th power)

Basic Input and Output :

- The programs that we will write request and get information from the user.
- In Python, the input function is used for this purpose,

```
>>> name = input('What is your name?: ')
```

- Characters within quotes are called strings.
- This particular use of a string, for requesting input from the user, is called a prompt.
- The input function displays the string on the screen to prompt the user for input,

>>>What is your name?: **Charles**

- The **underline** is used here to indicate the user's input.

- ***The print function*** is used to display information on the screen in Python.
- This may be used to display a message,

```
>>>  print('Welcome to My First Program!')

Welcome to My First Program!

>>> n  =  10

>>> print(n)

10
```

LITERALS

What Is a Literal?

- To take something literally is to take it at "face value."
- A literal is a sequence of one of more characters that stands for itself, such as the literal

Numeric Literals:

- A numeric literal is a literal containing only the digits 0–9, an optional sign character (+ or -), and a possible decimal point.
- (The letter **e** is also used in exponential notation, shown in the next subsection).
- If a numeric literal contains a decimal point, then it denotes a floating-point value, or "float" (e.g., 10.24); otherwise, it denotes an integer value (e.g., 10).
- ***Commas(,) are never used in numeric literals.***
- Figure 2-2 gives additional examples of numeric literals in Python.

Numeric Literals		
integer values	floating-point values	incorrect
5	5. 5.0 5.125 0.0005 5000.125	5,000.125
2500	2500. 2500.0 2500.125	2,500 2,500.125
+2500	+2500. +2500.0 +2500.125	+2,500 +2,500.125
-2500	-2500. -2500.0 -2500.125	-2,500 -2,500.125

FIGURE 2-2 Numeric Literals in Python

Limits of Range in Floating-Point Representation

- There is **no limit** to the size of an integer that can be represented in Python. Floating-point values, however, have both a limited range and a limited precision.
- Python uses a double-precision standard format (IEEE 754) providing a range of 10^{-308} to 10^{308} with 16 to 17 digits of precision.
- To denote such a range of values, floating-points can be represented in scientific notation,
- **Arithmetic overflow occurs when a calculated result is too large in magnitude to be represented.**
- **Arithmetic underflow occurs when a calculated result is too small in magnitude to be represented.**

Built-in format Function:

- Because floating-point values may contain an arbitrary number of decimal places, the built-in format function can be used to produce a numeric string version of the value containing a specific number of decimal places.

```
>>> 12/5
2.4
>>> format(12/5, '.2f')
'2.40'
```

```
>>> 5/7
0.7142857142857143
>>> format(5/7, '.2f')
'0.71'
```

- In these examples, format specifier **'.2f'** rounds the result to two decimal places of accuracy in the string produced. For very large (or very small) values 'e' can be used as a format specifier,

```
>>> format(2 ** 100, '.6e')
'1.267651e+30'
```

STRING LITERALS:

- Numerical values are not the only literal values in programming.
- String literals, or " strings ," represent a sequence of characters,

'Hello' 'Smith, John' "Baltimore, Maryland 21210"

- I n Python, string literals may be delimited (surrounded) by a matching pair of either **single (')** or **double (")** quotes.
- Strings must be contained all on one

>>> print('Welcome to Python!')

Welcome to Python!

```
'A'                          - a string consisting of a single character
'jsmith16@mycollege.edu'     - a string containing non-letter characters
"Jennifer Smith's Friend"    - a string containing a single quote character
' '                          - a string containing a single blank character
''                           - the empty string
```

FIGURE 2-3 String Literal Values

The Representation of Character Values:

- There needs to be a way to encode (represent) characters within a computer.
- Although various encoding schemes have been developed, the Unicode encoding scheme is intended to be a universal encoding scheme.
- Unicode is actually a collection of different encoding schemes utilizing between 8 and 32 bits for each character.
- The default encoding in Python uses UTF-8, an 8-bit encoding compatible with ASCII, an older, still widely used encoding scheme.
- Currently, there are over 100,000 Unicode-defined characters for many of the languages around the world.
- Unicode is capable of defining more than 4 billion characters.

Space	00100000	32	A	01000001	65
!	00100001	33	B	01000010	66
"	00100010	34	C	01000011	67
#	00100011	35	.		
.			.		
.			Z	01011010	90
0	00110000	48	a	01100001	97
1	00110001	49	b	01100010	98
2	00110010	50	c	01100011	99
.			.		
.			.		
9	00111001	57	z	01111010	122

FIGURE 2-4 Partial UTF-8 (ASCII) Code Table

CONTROL CHARACTERS:

- Control characters are special characters that are not displayed on the screen.
- Control characters do not have a corresponding keyboard character.
- Therefore, they are represented by a combination of characters called ***an escape sequence***.
- An escape sequence begins with an escape character that causes the sequence of characters following it to "escape" their normal meaning.
- **The backslash (\)** serves as the escape character in Python. For example, the escape sequence **'\n',** represents the newline control character , used to begin a new screen line.
- An example of its use is given below

```
>>>print('Hello\nJennifer Smith')
```

which is displayed as follows,

```
Hello  Jennifer Smith
```

String Formatting:

- W e saw above the use of built-in function format for controlling how numerical values are displayed.
- We now look at how the format function can be used to control how strings are displayed.
- As given above, the format function has the form,

format(value, format_specifi er)

Example:

```
>>>format('Hello', ' < 20')
→ 'Hello                '
>>>format('Hello', ' > 20')
→ '                Hello'
```

VARIABLES AND IDENTIFIERS:

What Is a Variable?

- A variable is a name (identifi er) that is associated with a value, as for variable num depicted in Figure 2-7.
- A variable can be assigned different values during a program's execution—hence, the name "variable."
- Wherever a variable appears in a program (except on the left-hand side of an assignment statement), it is the value associated with the variable that is used , and not the variable's name,
- num + 1 → 10 + 1 → 11 Variables are assigned values by use of the assignment operator , =,

```
num = 10
num = num + 1
```

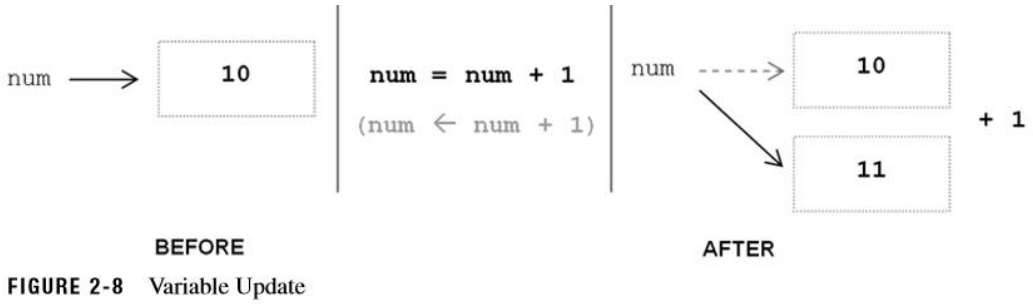

FIGURE 2-8 Variable Update

- A variable is a name that is associated with a value.
- The assignment operator, =, is used to assign values to variables.
- An immutable value is a value that cannot be changed.

Variable Assignment and Keyboard Input:

- The value that is assigned to a given variable does not have to be specified in the program, as demonstrated in previous examples.
- The value can come from the user by use of the input function.

```
>>> name = input('What is your first name?')

    What is your first name? John
```

- In this case, ***the variable name*** is assigned the string ***'John'.*** If the user hit return without entering any value, name would be assigned to the empty string (").
- All input is returned by the input function as a **string type.**
- For the input of numeric values, the response must be converted to the appropriate type.
- Python provides built-in type conversion functions **int ()** and **float()** for this purpose, as shown below for a **gpa** calculation program,

```
>>>line = input('How many credits do you have?')

>>>num_credits = int(line)

>>>line = input('What is your grade point average?')

>>>  = float(line)
```

- All input is returned by the input function as a string type.
- Built-in functions int() and float() can be used to convert a string to a numeric type.

What Is an Identifier?

- An identifier is a sequence of one or more characters used to provide a name for a given program element.
- Variable names line, **num_credits**, and **gpa** are each identifiers.

- Python is **case sensitive**, thus, **Line** is different from **line.**
- Identifiers may contain letters and digits, but cannot begin with a digit.
- The underscore character, _, is also allowed to aid in the readability of long identifier names.
- It should not be used as the first character, however, as identifiers beginning with an underscore have special meaning in Python.
- Spaces are not allowed as part of an identifier.
- This is a common error since some operating systems allow spaces within file names.

Valid Identifiers	Invalid Identifiers	Reason Invalid
totalSales	'totalSales'	quotes not allowed
totalsales	total sales	spaces not allowed
salesFor2010	2010Sales	cannot begin with a digit
sales_for_2010	_2010Sales	should not begin with an underscore

FIGURE 2-11 Identifier Naming

Keywords and Other Predefined Identifiers in Python:

- A keyword is an identifier that has predefined meaning in a programming language.
- Therefore, keywords cannot be used as "regular" identifiers.
- Doing so will result in a syntax error, as demonstrated in the attempted assignment to keyword and below,

```
>>> and = 10
SyntaxError: invalid syntax
```

Because, **and** is the reserved keywords.

- The keywords in Python are listed in Figure 2-12.

```
and       as        assert    break     class     continue  def
del       elif      else      except    finally   for       from
global    if        import    in        is        lambda    nonlocal
not       or        pass      raise     return    try       while
with      yield     false     none      true
```

FIGURE 2-12 Keywords in Python

OPERATORS

What Is an Operator?

- An operator is a symbol that represents an operation that may be performed on one or more operands.
- For example, the + symbol represents the operation of addition.
- An operand is a value that a given operator is applied to, such as operands 2 and 3 in the expression 2+3
- A unary operator operates on only one operand, such as the negation operator in -12.
- A binary operator operates on two operands, as with the addition operator.
- Most operators in programming languages are binary operators. We look at the arithmetic operators in Python next.

Arithmetic Operators:

Python provides the arithmetic operators given in Figure 2-15.

Arithmetic Operators		Example	Result
-x	negation	-10	-10
x + y	addition	10 + 25	35
x - y	subtraction	10 - 25	-15
x * y	multiplication	10 * 5	50
x / y	division	25 / 10	2.5
x // y	truncating div	25 // 10	2
		25 // 10.0	2.0
x % y	modulus	25 % 10	5
x ** y	exponentiation	10 ** 2	100

FIGURE 2-15 Arithmetic Operators in Python

- The +, -, * (multiplication) and / (division) arithmetic operators perform the usual operations.
- Note that the - symbol is used both as a unary operator (for negation) and a binary operator (for subtraction).
- Python also includes an exponentiation (**) operator.
- Integer and floating-point values can be used in both the base and the exponent,
- Python provides ***two forms of division***.
- **"true" division** is denoted by a single slash**, /**.
- Thus, 25 / 10 evaluates to 2.5.
- **Truncating division** is denoted by a double slash, //, providing a truncated result based on the type of operands applied to.
- When both operands are integer values, the result is a truncated integer referred to as integer division.

	Operands	result type	example	result
/ Division operator	int, int	float	`7 / 5`	`1.4`
	int, float	float	`7 / 5.0`	`1.4`
	float, float	float	`7.0 / 5.0`	`1.4`
// Truncating division operator	int, int	truncated int ("integer division")	`7 // 5`	`1`
	int, float	truncated float	`7 // 5.0`	`1.0`
	float, float	truncated float	`7.0 // 5.0`	`1.0`

FIGURE 2-16 Division Operators in Python

- The modulus operator (%) gives the remainder of the division of its operands, resulting in a cycle of values.

Modulo 7		Modulo 10		Modulo 100	
`0 % 7`	**0**	`0 % 10`	**0**	`0 % 100`	**0**
`1 % 7`	**1**	`1 % 10`	**1**	`1 % 100`	**1**
`2 % 7`	**2**	`2 % 10`	**2**	`2 % 100`	**2**
`3 % 7`	**3**	`3 % 10`	**3**	`3 % 100`	**3**
`4 % 7`	**4**	`4 % 10`	**4**	.	.
`5 % 7`	**5**	`5 % 10`	**5**	.	.
`6 % 7`	**6**	`6 % 10`	**6**	`96 % 100`	**96**
`7 % 7`	0	`7 % 10`	**7**	`97 % 100`	**97**
`8 % 7`	1	`8 % 10`	**8**	`98 % 100`	**98**
`9 % 7`	2	`9 % 10`	**9**	`99 % 100`	**99**
`10 % 7`	3	`10 % 10`	0	`100 % 100`	0
`11 % 7`	4	`11 % 10`	1	`101 % 100`	1
`12 % 7`	5	`12 % 10`	2	`102 % 100`	2

FIGURE 2-17 The Modulus Operator

RELATIONAL OPERATORS:

- The relational operators in Python perform the usual comparison operations, shown in Figure 3-3.

Relational Operators	Example	Result
== equal	10 == 10	True
!= not equal	10 != 10	False
< less than	10 < 20	True
> greater than	'Alan' > 'Brenda'	False
<= less than or equal to	10 <= 10	True
>= greater than or equal to	'A' >= 'D'	False

FIGURE 3-3 The Relational Operators

- Relational expressions are a type of Boolean expression, since they evaluate to a Boolean result. These operators not only apply to numeric values, but to any set of values that has an ordering, such as strings.
- Note the use of the comparison operator, = =, for determining if two values are equal.
- This, rather than the (single) equal sign, = , is used since the equal sign is used as the assignment operator.
- This is often a source of confusion for new programmers,

```
num = 10      variable num is assigned the value 10
num == 10     variable num is compared to the value 10
```

MEMBERSHIP OPERATORS:

- Python provides a convenient pair of membership operators.
- These operators can be used to easily determine if a particular value occurs within a specified list of values.
- The in operator is used to determine if a specific value is in a given list, returning **True if found**, and **False otherwise.**
- The **not in** operator returns the **opposite result**.
- The list of values surrounded by matching parentheses in the figure are called tuples in Python.
- The membership operators are given in Figure 3-4.

Membership Operators	Examples	Result
in	10 in (10, 20, 30)	True
	red in ('red','green','blue')	True
not in	10 not in (10, 20, 30)	False

FIGURE 3-4 The Membership Operators

Boolean Operators:

- Boolean algebra contains a set of Boolean (logical) operators, denoted by **and, or,** and **not** in Python.
- These logical operators can be used to construct arbitrarily complex Boolean expressions.
- The Boolean operators are shown in Figure 3-5.
- Logical and is true only when both its operands are true—otherwise, it is false.
- Logical or is true when either or both of its operands are true, and thus false only when both operands are false.
- Logical not simply reverses truth values—not False equals True, and not True equals False.

x	y	x and y	x or y	not x
False	False	False	False	True
True	False	False	True	False
False	True	False	True	
True	True	True	True	

FIGURE 3-5 Boolean Logic Truth Table

EXPRESSIONS AND DATA TYPES:

What Is an Expression?

- An expression is a combination of symbols that evaluates to a value.
- Expressions, most commonly, consist of a combination of operators and operands,

  4 + (3 * k)

- An expression can also consist of a single literal or variable.
- Thus, **4, 3,** and **k** are each expressions.
- This expression has two sub-expressions, **4** and **(3 * k).** Sub-expression **(3 * k)** itself has two sub-expressions, **3 and k.**
- Expressions that evaluate to a numeric type are called arithmetic expressions.

- A sub-expression is any expression that is part of a larger expression.

OPERATOR PRECEDENCE:

- The way we commonly represent expressions, in which operators appear between their operands, is referred to as infix notation.
- For example, the expression 4 + 3 is in infix notation since the + operator appears between its two operands, 4 and 3.
- There are other ways of representing expressions called prefix and postfix notation, in which operators are placed before and after their operands, respectively

Operator	Associativity
** (exponentiation)	right-to-left
- (negation)	left-to-right
* (mult), / (div), // (truncating div), % (modulo)	left-to-right
+ (addition), - (subtraction)	left-to-right

FIGURE 2-19 Operator Precedence of Arithmetic Operators in Python

OPERATOR PRECEDENCE AND BOOLEAN EXPRESSIONS:

- The operator precedence (and operator associativity) of arithmetic operators was given in Chapter 2.
- Operator precedence also applies to Boolean operators.
- Since Boolean expressions can contain arithmetic as well as relational and Boolean operators, the precedence of all operators needs to be collectively applied.
- An updated operator precedence table is given in Figure 3-6.

Operator	Associativity
** (exponentiation)	right-to-left
- (negation)	left-to-right
* (mult), / (div), // (truncating div), % (modulo)	left-to-right
+ (addition), - (subtraction)	left-to-right
<, >, <=, >=, !=, == (relational operators)	left-to-right
not	left-to-right
and	left-to-right
or	left-to-right

FIGURE 3-6 Operator Precedence of Arithmetic, Relational, and Boolean Operators

As before, in the table, higher-priority operators are placed above lower-priority ones. Thus, we see that *all arithmetic operators are performed before any relational or Boolean operator*,

```
10 + 20 < 20 + 30     →  30 < 50 → True
```

In addition, *all of the relational operators are performed before any Boolean operator*,

```
10 < 20 and 30 < 20   →  True and False → False
10 < 20 or 30 < 20    →  True or False → True
```

And as with arithmetic operators, Boolean operators have various levels of precedence. Unary Boolean operator `not` has higher precedence than `and`, and Boolean operator `and` has higher precedence than `or`.

```
10 < 20 and 30 < 20 or 30 < 40  →  True and False or True
                                →  False or True → True

not 10 < 20 or 30 < 20          →  not True or False
                                →  False or False → False
```

As with arithmetic expressions, it is good programming practice to use parentheses, even if not needed, to add clarity and enhance readability. Thus, the above expressions would be better written by denoting at least some of the subexpressions,

```
(10 < 20 and 30 < 20) or (30 < 40)
(not 10 < 20) or (30 < 20)
```

if not all subexpressions,

```
((10 < 20) and (30 < 20)) or (30 < 40)
(not (10 < 20)) or (30 < 20)
```

Finally, note from Figure 3.6 above that all relational and Boolean operators associate from left to right.

OPERATOR ASSOCIATIVITY:

- A question that you may have already had is, "What if two operators have the same level of precedence, which one is applied first?"
- For operators following the associative law, the order of evaluation doesn't matter,

```
(2 + 3) + 4 → 9      2 + (3 + 4) → 9
```

- In this case, we get the same results regardless of the order that the operators are applied.

- Division and subtraction, however, do not follow the associative law,

```
(a) (8 - 4) - 2 → 4 - 2 → 2        8 - (4 - 2) → 8 - 2 → 6
(b) (8 / 4) / 2  → 2 / 2 → 1       8 / (4 / 2)  → 8 / 2 → 4
(c) 2 ** (3 ** 2) → 512            (2 ** 3) ** 2 → 64
```

- Here, the order of evaluation does matter.
- To resolve the ambiguity, each operator has a specified operator associativity that defines the order that it and other operators with the same level of precedence are applied

Operator	Associativity
** (exponentiation)	right-to-left
- (negation)	left-to-right
* (mult), / (div), // (truncating div), % (modulo)	left-to-right
+ (addition), - (subtraction)	left-to-right

FIGURE 2-19 Operator Precedence of Arithmetic Operators in Python

Mixed-Type Expression

- A mixed-type expression is an expression containing operands of different type.
- The CPU can only perform operations on values with the same internal representation scheme, and thus only on operands of the same type.
- Operands of mixed-type expressions therefore must be converted to a common type.
- Values can be converted in one of two ways—by implicit (automatic) conversion, called coercion, or by explicit type conversion.

Coercion vs. Type Conversion:

- **Coercion** is the implicit (automatic) conversion of operands to a common type.
- Coercion is automatically performed on mixed-type expressions only if the operands can be safely converted, that is, if no loss of information will result.
- The conversion of integer 2 to floating-point **2.0** below is a safe conversion—the conversion of **4.5** to integer **4** is not, since the decimal digit would be lost,

```
2 + 4.5 → 2.0 + 4.5 → 6.5 safe (automatic conversion of int to float)
```

Type conversion:

- Type conversion is the explicit conversion of operands to a specific type.
- Type conversion can be applied even if loss of information results.

- Python provides built-in type ***conversion functions int() and float(),*** with the ***int() function*** truncating results as given in Figure 2-21.

```
float(2) + 4.5  →  2.0 + 4.5  →  6.5
2 + int(4.5)  →  2 + 4  →  6
```

Conversion Function		Converted Result		Conversion Function		Converted Result
int()	int(10.8)	10		**float()**	float(10)	10.0
	int('10')	10			float('10')	10.0
	int('10.8')	ERROR			float('10.8')	10.8

FIGURE 2-21 Conversion Functions int() and float() in Python