\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CSC121 PYTHON Programming**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lesson 02 Writing Simple Python Programs

# **Objectives**

In this lesson, students will learn:

- To use variables to store data

- To write statements to get input

- To write statements to perform calculations

- To write statements to display output

# **2.1 Introduction**

In Lesson 01 we learned how to design algorithms for simple computer programs. Algorithms we have seen so far include three types of steps:

1. Input steps: Inputting data from user
2. Processing steps: Using input data to perform calculations
3. Output steps: Displaying results of calculations

In this lesson, we will learn how to write Python code to implement these steps.

# **2.2 Variables**

Before we look at how to write Python statements to get input, perform calculations and display output, let’s start with a very important concept: variables.

Most computer programs need to work with data. Many programs start with asking user to enter input. For example, a program that calculates the sum of two numbers may ask the user to enter two numbers when the program begins. The two numbers entered by the user are data. The program must store the input values in computer memory so they can be used later in the program.

In addition to user input, anything calculated in a step usually needs to be stored in memory, too. A program step that calculates the sum of two numbers also needs memory space to store the sum. If the sum is not stored in memory, it will be unavailable when the program tries to display or use it later.

Inside a Python program, we use something called **variables** to represent data stored in computer memory. Each variable in a program is associated with a cell in the computer’s memory. The data item stored in the memory cell is called the **value** of the variable.

Every variable in a simple Python program must have a unique name. The program uses the variable name to tell the computer which memory location it wants to access. We usually make up a meaningful name so we can tell what is stored in that memory cell from the name. For example, we may use the name sum for the variable that stores the sum of two numbers.

We must follow some rules when we make up a name for a variable:

1. A name consists of a series of characters, but not all characters can be used in a name. Only letters, digits, and the underscore symbol can be used. For example, the names ssn and s\_s\_n are valid but the name s.s.n. is invalid because periods are not allowed.
2. The first character of a name can be a letter or underscore, but it cannot be a digit. For example, the name firstTest is valid but the name 1stTest is invalid because it starts with a digit.
3. Uppercase and lowercase letters are different. For example, height and Height are considered two different names because the letter H is in lowercase in the first name but in uppercase in the second name.
4. Python has a set of keywords. Every keyword has a special meaning. We cannot name a variable which is exactly the same as one of the keywords. For example, the word raise is a keyword in Python. Therefore, we cannot name a variable as raise.
5. We cannot have space in a name. For example, the name midterm exam score is invalid because there are spaces between words. If we need a variable to store midterm exam score, we may name it midtermexamscore, midterm\_exam\_score, or midtermExamScore. Although all three are valid, programmers prefer the second and third ways because they make it easier to see the words in the variable name.

# **2.3 Writing Statements to Store Data**

In a Python program, we use assignment statements to store data in computer memory. The following is an example of assignment statements:

age = 16

When the computer executes this statement, it stores 16 in a memory cell. Each memory cell has a unique address. The computer uses memory address to keep track of where a data item is stored. Suppose 16 is stored in the memory cell with the address 501462. Later when the program needs to retrieve this data item, it must go to the memory cell 501462 to retrieve 16 from there.

The program does not need to know the address of the memory cell that stores the data item. The left hand side of the assignment statement creates a variable named age. The computer will associate this variable with the memory address. Later when we need to access the data item, we can use the variable name to access it. Look at the following statement:

age\_next\_yr = age + 1

The computer will do three things when it executes this assignment statement:

1. Retrieves 16 from the memory cell associated with the variable age
2. Calculates 16 + 1
3. Store the result 17 in a memory cell and associate the memory cell with the variable age\_next\_yr. Programmers typically simply say “store 17 in variable age\_next\_yr”.

Let’s look at a few more examples of assignment statements.

The following statement stores 3.52 in the variable gpa:

gpa = 3.52

The following statements stores 25000 in salary and 28000 in new\_salary:

salary = 25000

new\_salary = 28000

Suppose later in the program we have the following statement:

salary\_change = new\_salary - salary

The computer will retrieve 28000 from new\_salary and 25000 from salary, calculate the difference and store 3000 in the variable salary\_change.

If a statement stores a new value in a variable that already exists, the new value will overwrite the old value. Let’s look at the following four statements:

midterm\_score = 80

final\_score = 75

final\_score = 85

total\_score = midterm\_score + final\_score

The first statement stores 80 in midterm\_score. The second statement stores 75 in final\_score. The third statement stores a new value 85 in final score to replace its old value. When the fourth statement executes, 80 is retrieved from midterm\_score while 85 is retrieved from final\_score. The sum 165 is stored in total\_score.

Although it is not necessary, it is good to know how Python deals with computer memory when the value of a variable is changed because Python is different from many other programming languages in this. Other programing languages simply store a new value to overwrite the old value in the memory cell associated with the variable. For example, suppose memory cell 524617 is associated with the variable final\_score and is storing 75 currently. The new value 85 will be stored in the same memory cell to replace the old value. Python handles this in a different way. It actually stores 85 in a new memory cell (let’s say its address is 524700) and associate the variable to finlal\_score to the new memory cell.

Memory cell 524700

Data stored: 85

Memory cell 524617

Data stored: 75

final\_score

Since final\_score is now associated with memory cell 524700 instead of 524617, its value is 85 instead of 75. The memory cell 524617 will be reclaimed and reused by the operating system since it is not associated with any variable anymore.

Let’s stop a while and introduce a few technical terms. A literal is something in the program code that should be taken at its “face value”. For example, 16 in the following statement is a literal because it simply means the numerical value 16:

age = 16

Similarly, 3.52 in the following statement is a literal because it simply means the numerical value 3.52:

gpa = 3.52

In the opposite, age and gpa are not literals because they are variables. Their values depend on what are stored in the memory cells associated with them. Please read textbook sections 2.1.1 and 2.1.2 for more about literals.

Operators are symbols that represent operations that may be performed on one or more operands. For example, + is an operator while midterm\_score and final\_score are operands in the following statement:

total\_score = midterm\_score + final\_score

Similarly, - is an operator while new\_salary and salary are operands in the following statement:

salary\_change = new\_salary - salary

Please read textbook sections 2.3 for more about operators.

An expression is a combination of symbols that evaluates to a value. For example, in the following statement, midterm\_score + final\_score is an expression because it evaluates to a value after the addition of midterm\_score and final\_score is computed:

total\_score = midterm\_score + final\_score

Similarly, new\_salary - salary is an expression in the following statement

salary\_change = new\_salary - salary

Please read textbook sections 2.4.1, 2.4.2 and 2.4.3 for more about expressions.

In addition to storing numbers, we can also use assignment statements to store text in a variable. The following is an example:

college = 'Wake Tech'

Texts are called strings in programming. The string literal Wake Tech is stored in the variable college. It is a literal because it just represents the sequence of characters Wake Tech itself. It is not used a variable name or used in the program to represent anything else. It just simply represents the sequence of 9 characters: W is the first character, h is the last. The fifth character is a space character, which is allowed in string literals.

When you write Python statements, you must enclose string literals in quotes. In fact, the Python interpreter uses the quotes to tell Wake Tech is a string literal. Without the quotes, the Python interpreter will get confused when it read the statement:

college = Wake Tech

It will mistakenly view Wake Tech as the name of a variable and give you a syntax error.

In addition to single quotes, you can choose to enclose string literals in double quotes if you want to. Example:

college = "Wake Tech"

This statement is the same as the previous statement that uses single quotes. Both statements stores the 9-character string literal Wake Tech in the variable college. Python allows us to use either single quotes or double quotes to avoid possible confusions. Look at the following example,

movie\_name = "Sophie's choice"

This statement will successfully store the 15-character literal in the variable movie\_name, but the following statement will not:

movie\_name = 'Sophie's choice'

The problem with the statement that uses single quotes is that the Python interpreter will mistakenly view the apostrophe between the letter e and s as a quote and think you have syntax error in the statement.

Please read textbook section 2.1.3 for more about string literals.

# **2.4 Displaying Output**

We need statements to display output in a program. The computer will not display any output if the program does not tell it to do so. A program is pretty useless if the user does not see any output.

In Python, we use print functions to display output. The following is an example,

print('This is my first program')

It tells the computer to display the following text:

This is my first program

The string literal “This is my first program” is displayed because this string literal is placed inside the parentheses of the print function. Similarly, the statement

print('Programming is fun!')

displays the following output:

Programming is fun!

If we place the name of a variable inside the parentheses, the computer will display the value of the variable. See this example:

age = 16  
age\_next\_yr = age + 1  
print(age\_next\_yr)

The program above displays 17, which is the value stored in the variable age\_next\_yr.

The user may get confused if we only display the value of age\_next\_yr. we should add a little description to explain what that output is:

age = 16  
age\_next\_yr = age + 1  
print(**'Age next year:'**)  
print(age\_next\_yr)

The following is the output of the program:

Age next year:

17

We can shorten the program by combining the two print functions into one:

age = 16  
age\_next\_yr = age + 1  
print(**'Age next year:'**, age\_next\_yr)

The following is the new output:

Age next year: 17

You can display as many items as you want with one single print function. Simply place all the items you want to display inside the parentheses and separate them with commas. When those items are display, a space will be inserted between them by default. Example:

age = 16  
age\_next\_yr = age + 1  
print(**'You are'**, age\_next\_yr, **'years old next year.'**)

The print function displays three items: the string ‘You are’, the value of age\_next\_yr, and the string ‘years old next year’. The following is the new output:

You are 17 years old next year.

Let’s see another example.

salary = 25000  
new\_salary = 28000  
salary\_change = new\_salary - salary  
print(**'You are going to earn $'**, salary\_change, **'more per year.'**)

The following is the new output:

You are going to earn $ 3000 more per year.

# **2.5 Getting Input**

The age program we saw earlier is quite useless because it only calculates new age of persons who are 16 years old this year. It cannot be used to for people who are not 16 years old. Similarly, salary change program can be used only for people who are making $25000 now and $28000 next year. What is missing from these programs is the ability for the user to enter data into the program when the program is running.

In Python, we use the input function to input data from the keyboard. If we put the input function on the right hand side of an assignment statement, the computer will read user input from the keyboard and store it in the variable on the right hand side. Example:

age = input()

The statement above stores whatever entered by the user in the variable age. We need to display some text on the screen so the user will know what to enter. We can insert a string inside the input function, which will display the string before it waits for the user to enter input:

age = input(**'How old are you now? '**)

When the computer execute this statement, it will show ‘How old are you now?’ on the screen and then wait for the user to enter age:

How old are you now?

Similarly, we can use the following two statements to get current and new salaries and store them in the variables salary and new\_salary:

salary = input(**'Enter your current salary: '**)  
new\_salary = input(**'Enter your new salary: '**)

Let’s add a statement to get age in the age program:

age = input(**'How old are you now? '**)  
age\_next\_yr = 1 + age  
print(**'You are'**, age\_next\_yr, **'years old next year.'**)

Let’s run the new program and enter 20:

How old are you now? 20

Instead of calculating and displaying age next year, the program actually gets the following error message:

Traceback (most recent call last):

File "C:/ageExample.py", line 2, in <module>

age\_next\_yr = 1 + age

TypeError: unsupported operand type(s) for +: 'int' and 'str'

To understand what went wrong, we must spend some time to discuss an important topic about data. The issue here is data type.

Programs need to deal with different types of data. The most common types include text, integers and real numbers (i.e. numbers with decimal point). Different types of data are stored in computer memory in different ways, and some operations may be inappropriate for some types. For example, it makes sense to divide a number by 2.5, but it does not make sense to divide the text ‘Hello’ by 2.5.

Python has a few built-in data types to store texts and numbers. When text, which is a sequence of characters, is stored in a variable, its type is string. For example, when ‘Wake Tech’ is stored in a variable, its type is string and it is encoded in computer memory in a way specially designed to store characters.

Python has two types for numerical values: integers and floating point numbers. Integers are whole numbers such as 17 and -25. Floating point numbers are number with decimal point such as 3.27 and -16.25. Integers and floating point numbers are encoded in computer memory in very different ways. Exactly how they are encoded is beyond the scope of this course.

When we use an assignment statement to store a value in a variable, Python will determine the type automatically by looking at what is being stored. For example, the following statement stores 16 in the variable age:

age = 16

Since 16 is a whole number, Python will store 16 in the variable age as an integer. That means in the memory cell associated with age, 16 is encoded in the format designed for whole numbers.

Similarly, 80 is stored as an integer in the variable midterm\_score by the following statement:

midterm\_score = 80

The following statement stores 3.52 as a floating point number in the variable gpa because 3.52 has a decimal point:

gpa = 3.52

Similarly, 17.5 will be stored as a float point number in the variable age:

age = 17.5

Look at the following statement. What type will be used to store 17.0?

age = 17.0

Since 17.0 has a decimal point in it, it will be stored as a floating point number.

Let’s look at the string type. The following example stores Wake Tech in the variable college using string type:

college = 'Wake Tech'

The reason Wake Tech is stored as a string is the quotes. The quotes tell Python that Wake Tech is a string literal and it should be encoded in memory in the format designed for string of characters. Without the quotes, Python will give you a syntax error.

Similarly, CSC121 is stored in the variable course as a string:

course = 'CSC121'

Look at the following statement. What type will be used to store 17?

age = '17'

17 will be stored in the variable age as a string because the quotes make Python views 17 as a two-character string instead of an integer.

What’s wrong if 17 is stored as a string instead of an integer? Nothing is wrong until we try to add 1 to age. Look at the following example:

age = **'17'**age\_next\_yr = 1 + age  
print(**'You are'**, age\_next\_yr, **'years old next year.'**)

Python cannot add 1 to age because age is not a number. The following is the error message:

Traceback (most recent call last):

File "C:/ageExample.py", line 2, in <module>

age\_next\_yr = 1 + age

TypeError: unsupported operand type(s) for +: 'int' and 'str'

This is exactly the same error message we got earlier when we input age from the user. Let’s look at that program again:

age = input(**'How old are you now? '**)  
age\_next\_yr = 1 + age  
print(**'You are'**, age\_next\_yr, **'years old next year.'**)

We got the same error message when we type 17 and hit Enter in the keyboard. The error message says it cannot perform addition with an integer and a string. What does it tell you? 1 is an integer. That means what is stored in the variable age is a string, not an integer! Why?

The reason is that when we use the input function to get user input, it records the sequence of keys typed in the keyboard. When it is stored in the variable on the left hand side of an assignment statement, it is stored as a sequence of characters, i.e. a string. Therefore, when the user type 1 and 7 on the keyboard, it is not recorded and stored as a numerical value. Instead, it is recorded and stored as a 2-character string.

Now we know that 17 is stored as a string instead of a numerical value in age. What can we do to fix the problem? Fortunately there is a way to convert a string to a numerical value. Look at the following code:

age = input(**'How old are you now? '**)  
age = int(age)  
age\_next\_yr = 1 + age  
print(**'You are'**, age\_next\_yr, **'years old next year.'**)

We added a new statement

age = int(age)

Right after we got age from the user. What does this statement do? The right hand side is an int function. What it does is finding the equivalent integer value of the string age. For example, if the string '17' is stored in age, its equivalent integer value is the integer 17. This integer is stored in the variable on the left hand side of the assignment statement. Since the variable on the left hand side is the same variable age, the integer 17 will be stored in age to replace the string '17'.

Memory cell 524617

Data stored: '17'

age

Memory cell 524700

Data stored: 17

Suppose age is originally associated with the memory cell 524617, which have the string '17' stored there. Now the int function generates the integer 17 and stores it in memory cell 524700. The variable age cuts its association with memory cell 524617 and establish association with memory cell 524700. When the computer executes the next statement:

age\_next\_yr = 1 + age

The integer 17 is retrieved from memory cell 524700 and used in the addition.

Here is the salary example:

salary = input(**'Enter your current salary: '**)  
salary = int(salary)  
new\_salary = input(**'Enter your new salary: '**)  
new\_salary = int(new\_salary)  
salary\_change = new\_salary - salary  
print(**'You are going to earn $'**, salary\_change, **'more per year.'**)

We use the int function to convert the input strings into integers.

Sometimes we need to convert input strings into floating point numbers. The float function, which is similar to the int function, can used to achieve this goal. The following is an example:

gpa\_last\_sem = input(**'What was your GPA last semetser? '**)  
gpa\_last\_sem = float(gpa\_last\_sem)  
gpa\_this\_sem = input(**'What is your GPA this semetser? '**)  
gpa\_this\_sem = float(gpa\_this\_sem)  
gpa\_change = gpa\_this\_sem - gpa\_last\_sem  
print(**'GPA change:'**, gpa\_change)

Since we expect the user to enter numbers with decimal points, we need to use the float function to convert the input strings into floating pint numbers.

If you are not sure whether the user will enter integers or floating point numbers, you should use float. It is okay to store a whole number as a floating point value. For example, we can use the float function in the age program:

age = input(**'How old are you now? '**)  
age = float(age)  
age\_next\_yr = 1 + age  
print(**'You are'**, age\_next\_yr, **'years old next year.'**)

Suppose the user enters 17. The input string, which is stored in the variable age by the first statement, will be converted to the floating point number 17.0 and stored in age by the second statement. 1 + 17.0 is 18.0. Therefore 18.0 is stored in age\_next\_yr by the third statement and displayed in the console window by the fourth statement.

How old are you now? 17

You are 18.0 years old next year.

Similarly, we can use the float function in the salary program:

salary = input(**'Enter your current salary: '**)  
salary = float(salary)  
new\_salary = input(**'Enter your new salary: '**)  
new\_salary = float(new\_salary)  
salary\_change = new\_salary - salary  
print(**'You are going to earn $'**, salary\_change, **'more per year.'**)

The following is a sample test run of the program:

Enter your current salary: 30000

Enter your new salary: 35000

You are going to earn $ 5000.0 more per year.

One last thing before we end this discussion. In the last several examples we use one statement to store input string and another statement to convert it into a numerical value. You can combine the two statements into one. For example,

age = input(**'How old are you now? '**)  
age = float(age)

can be combined into one statement:

age = float(input(**'How old are you now? '**))

What we did was directly put the input function inside the parentheses of the float function to replace the variable age. That means as soon as an input string is read from the keyboard by the input function, it is sent to the float function immediately to find its equivalent floating point value, which will be stored in the variable age. That means the input string was never saved in the variable age. As soon as the variable age is created, a floating point value is saved there. The following is the complete program code:

age = float(input(**'How old are you now? '**))  
age\_next\_yr = 1 + age  
print(**'You are'**, age\_next\_yr, **'years old next year.'**)

The following is the new program code for the salary program:

salary = float(input(**'Enter your current salary: '**))  
new\_salary = float(input(**'Enter your new salary: '**))  
salary\_change = new\_salary - salary  
print(**'You are going to earn $'**, salary\_change, **'more per year.'**)

Let’s look at one more example before we end this discussion. The following program calculates the average of midterm and final exam scores:

midterm\_score = float(input(**'Enter midterm score: '**))  
final\_score = float(input(**'Enter final score: '**))  
avg\_score = (midterm\_score + final\_score)/2  
print(**'Your average score:'**, avg\_score)

The following is a sample test run:

Enter midterm score: 89

Enter final score: 92

Your average score: 90.5

# **2.6 Further Reading**

Please chapter 2 of the textbook. Section 2.1 introduces numerical and string literals. It talks about range and precision issues of floating point numbers, and the way to control how numerical values and strings look when they are displayed. Section 2.2 introduces variables and how to store data in them. Sections 2.3 and 2.4 introduce operators, expressions and data types. Section 2.5 uses an age in seconds program to conclude the whole chapter.