CSC101 – Introduction to ICT Elementary Programming

Writing a Simple Program

Writing a program involves designing a strategy for solving the problem and then using a programming language to implement that strategy.

- Let's first consider the simple *problem* of computing the area of a circle.
  - How do we write a program for solving this problem?
- Writing a program involves designing algorithms and then translating them into programming instructions, or code.
- An *algorithm* describes how a problem is solved by listing the actions that need to be taken and the order of their execution.
- The algorithm for calculating the area of a circle can be described as follows:

1. Get the circle's radius from the user.
2. Compute the area by applying the following formula:

$$area = radius \times radius \times \pi$$

3. Display the result.

Writing a Simple Program

Writing a program involves designing a strategy for solving the problem and then using a programming language to implement that strategy.

- The value for the radius is stored in the computer's memory.
- In order to access it, the program needs to use a *variable*.
- A variable is a name that references a value stored in the computer's memory.
  - Rather than using **x** and **y** as variable names, choose *descriptive names*:
    - in this case, for example, you can use the name **radius** for the variable that references a value for radius and **area** for the variable that references a value for area.

Writing a Simple Program

Writing a program involves designing a strategy for solving the problem and then using a programming language to implement that strategy.

- The second step is to compute **area** by assigning the result of the expression
  - **radius * radius * 3.14159** to **area**.
- In the final step, the program will display the value of **area** on the console by using Python's **print** function.

Writing a Simple Program

```
1  # Assign a value to radius
2  radius = 20 # radius is now 20
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius", radius, "is", area)
```

radius → 20

area → 1256.636

```
The area for the circle of radius 20 is 1256.636
```

File Name: ComputeArea.py

Reading Input from the Console

Reading input from the console enables the program to accept input from the user.

variable = input(**"Enter a value: "**)

- The value entered is a string.
- You can use the function **eval** to evaluate and convert it to a numeric value.
  - For example,
    - **eval("34.5")** returns **34.5**,
    - **eval("345")** returns **345**,
    - **eval("3 + 4")** returns **7**, and
    - **eval("51 + (54 * (3 + 2))")** returns **321**.

Reading Input from the Console: Example

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius", radius, "is", area)
```

```
Enter a value for radius: 2.5 Enter
The area for the circle of radius 2.5 is 19.6349375
```

```
Enter a value for radius: 23 Enter
The area for the circle of radius 23 is 1661.90111
```

File Name: ComputeAreaWithConsoleInput.py

Reading Input from the Console: Example

```
1  # Prompt the user to enter three numbers
2  number1 = eval(input("Enter the first number: "))
3  number2 = eval(input("Enter the second number: "))
4  number3 = eval(input("Enter the third number: "))
5
6  # Compute average
7  average = (number1 + number2 + number3) / 3
8
9  # Display result
10 print("The average of", number1, number2, number3,
11     "is", average)
```

```
Enter the first number: 1 Enter
Enter the second number: 2 Enter
Enter the third number: 3 Enter
The average of 1  2  3 is 2.0
```

File Name: ComputeAverage.py

Identifiers

Identifiers are the names that identify the elements such as variables and functions in a program.

- In programming terminology, names assigned to variables, functions and labels are called *identifiers*.
- All identifiers must obey the following rules:
  - An identifier is a sequence of characters that consists of letters, digits, and underscores (_).
  - An identifier must start with a letter or an underscore. It cannot start with a digit.
  - An identifier cannot be a keyword. *Keywords*, also called *reserved words*, have special meanings in Python.
    - For example, **import** is a keyword, which tells the Python interpreter to import a module to the program.
  - An identifier can be of any length.

Variables, Assignment Statements, and Expressions

- *Variables are used to reference values that may be changed in the program.*
- The statement for assigning a value to a variable is called an *assignment statement*.
  - In Python, the equal sign (**=**) is used as the *assignment operator*.
  - The syntax for assignment statements is as follows:
    - variable = expression
- An *expression* represents a computation involving values, variables, and operators that, taken together, evaluate to a value. For example, consider the following code:
  - y = **1** # Assign 1 to variable y
  - radius = **1.0** # Assign 1.0 to variable radius
  - x = **5** * (**3** / **2**) + **3** * **2** # Assign the value of the expression to x

Simultaneous Assignments

- Python also supports *simultaneous assignment* in syntax like this:
  - var1, var2, ..., varn = exp1, exp2, ..., expn

```
1  # Prompt the user to enter three numbers
2  number1, number2, number3 = eval(input(
3      "Enter three numbers separated by commas: "))
4
5  # Compute average
6  average = (number1 + number2 + number3) / 3
7
8  # Display result
9  print("The average of", number1, number2, number3
10     "is", average)
```

```
Enter three numbers separated by commas: 1, 2, 3 Enter
The average of 1  2  3 is 2.0
```

File Name: ComputeAverageWithSimultaneousAssignment.py

Named Constants

A named constant is an identifier that represents a permanent value.

- The value of a variable may change during the execution of a program, but a *named constant* (or simply *constant*) represents permanent data that never changes.
- In our **ComputeArea** program, π is a constant.
  - If you use it frequently, you don't want to keep typing **3.14159**; instead, you can use a descriptive name **PI** for the value.
- Python does not have a special syntax for naming constants.
- You can simply create a variable to denote a constant.

Named Constants

- There are three benefits of using constants:

1. You don't have to repeatedly type the same value if it is used multiple times.
2. If you have to change the constant's value (e.g., from **3.14** to **3.14159** for **PI**), you need to change it only in a single location in the source code.
3. Descriptive names make the program easy to read.

Numeric Data Types and Operators [1]

- *Python has two numeric types—integers and* floating-point numbers*—for working with the operators* **+**, **-**, *****, **/**, **//**, ******, *and* **%**.
- The information stored in a computer is generally referred to as *data*.
  - There are two types of numeric data: integers and real numbers.
- Integer types (*int* for short) are for representing whole numbers.
- Real types are for representing numbers with a fractional part.
  - Real numbers are represented as floating-point (or *float*) values.

Numeric Data Types and Operators [2]

- The operators for numeric data types include the standard arithmetic operators, as shown in Table 2.1. The *operands* are the values operated by an operator.

TABLE 2.1 Numeric Operators

Name	*Meaning*	*Example*	*Result*
+	Addition	`34 + 1`	`35`
-	Subtraction	`34.0 - 0.1`	`33.9`
*	Multiplication	`300 * 30`	`9000`
/	Float Division	`1 / 2`	`0.5`
//	Integer Division	`1 // 2`	`0`
**	Exponentiation	`4 ** 0.5`	`2.0`
%	Remainder	`20 % 3`	`2`

Numeric Data Types and Operators [3]

The / operator performs a float division that results in a floating number

```
>>> 4 / 2
2.0
>>> 2 / 4
0.5
>>>
```

The // operator performs an integer division; the result is an integer, and any fractional part is truncated.

```
>>> 5 // 2
2
>>> 2 // 4
0
>>>
```

The % operator, known as *remainder* or *modulo* operator, yields the remainder after division.

```
1  # Prompt the user for input
2  seconds = eval(input("Enter an integer for seconds: "))
3
4  # Get minutes and remaining seconds
5  minutes = seconds // 60      # Find minutes in seconds
6  remainingSeconds = seconds % 60   # Seconds remaining
7  print(seconds, "seconds is", minutes,
8      "minutes and", remainingSeconds, "seconds")
```

```
Enter an integer for seconds: 500 Enter
500 seconds is 8 minutes and 20 seconds
```

Evaluating Expressions and Operator Precedence

- *Python expressions are evaluated in the same way as arithmetic expressions.*

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

can be translated into a Python expression as:

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9 * (4 / x + (9 + x) / y)
```

Evaluating Expressions and Operator Precedence

1. Exponentiation (**) is applied first.
2. Multiplication (*), float division (/), integer division (//) , and remainder operators (%) are applied next.
   - If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
3. Addition (+) and subtraction (-) operators are applied last.
   - If an expression contains several addition and subtraction operators, they are applied from left to right.

Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

Type Conversions and Rounding

If one of the operands for the numeric operators is a float value, the result will be a float value.

- Can you perform binary operations with two operands of different types?
- Yes. If an integer and a float are involved in a binary operation, Python automatically converts the integer to a float value.
- This is called *type conversion*.

- So, **3 * 4.5** is the same as **3.0 * 4.5**.

Type Conversions and Rounding

```
>>> value = 5.6
>>> int(value)
5
>>>
```

```
>>> value = 5.6
>>> round(value)
6
>>>
```

```
>>> value = 5.6
>>> round(value)
6
>>> value
5.6
>>>
```

```
1  # Prompt the user for input
2  purchaseAmount = eval(input("Enter purchase amount: "))
3
4  # Compute sales tax
5  tax = purchaseAmount * 0.06
6
7  # Display tax amount with two digits after decimal point
8  print("Sales tax is", int(tax * 100) / 100.0)
```

```
Enter purchase amount: 197.55 Enter
Sales tax is 11.85
```