Expressions

Expressions

Eric Roberts
CSCI 121
August 29, 2018

Data Types

- To a large extent, computers are used to process data that can take many forms. The diversity of possible data values gives rise to the notion of a ***data type***, which defines the common characteristics of data values that have a particular form or purpose.
- In computer science, each data type is defined by a ***domain***, which is the set of values that belong to that type, and a ***set of operations***, which shows how the values in that domain can be manipulated.
- Python includes three data types for numbers: the data type `int` for ***integers*** or whole numbers, the data type `float` for ***floating-point numbers*** containing a decimal point, and the data type `complex` for ***complex numbers,*** which are beyond the scope of this course.

Arithmetic Expressions

- Like most languages, Python specifies computation in the form of an ***arithmetic expression***, which consists of ***terms*** joined together by ***operators***.
- Each term in an arithmetic expression is one of the following:
  - An explicit numeric value, such as 2 or 3.14159265
  - A variable name that serves as a placeholder for a value
  - A function call that computes a value
  - An expression enclosed in parentheses
- The operators include the conventional ones from arithmetic, along with a few that are somewhat less familiar:

`+`	Addition	`//`	Quotient (floor division)
`-`	Subtraction	`%`	Remainder
`*`	Multiplication	`**`	Exponentiation
`/`	Division (exact)		

Division and Remainder

- Python has three operators that involve division:
  - The `/` operator computes the exact result of division, so that the expression `6 / 4` has the value 1.5. Applying the `/` operator always produces a floating-point value.
  - The `//` operator implements ***floor division,*** which is the result of exact division rounded down to the next smaller integer. The expression `6 // 4` produces the value 1, and `-6 // 4` produces the value –2.
  - The `%` operator implements what mathematicians call the mod operator. The reader describes this operator in detail, but it is easiest to think of as computing the remainder, at least for positive values. For example, `6 % 4` produces the value 2.
- The `%` operator turns out to be useful in a surprising number of programming applications and is well worth a bit of study.

Using the IDLE Interpreter

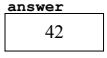
- The easiest way to get a sense of how arithmetic expressions work is to enter them into the IDLE interpreter:

IDLE

```
>>> 2 + 2
4
>>> 342 - 173
169
>>> 12345679 * 63
777777777
>>> 9 * 9 * 9 + 10 * 10 * 10
1729
>>>
```

Variables

- The simplest terms that appear in expressions are constant literals and variables. A ***variable*** is a placeholder for a value that can be updated as the program runs.
- A variable in Python is most easily envisioned as a box capable of storing a value



- Each variable has the following attributes:
  - A ***name***, which enables you to tell the variables apart.
  - A ***value***, which represents the current contents of the variable.
- The name of a variable is fixed; the value changes whenever you ***assign*** a new value to the variable.

Assignment

- In Python, you create a variable simply by assigning it a value in an ***assignment statement***, which has the general form:

  variable `=` *expression*

- The effect of an assignment statement is to compute the value of the expression on the right side of the equal sign and assign that value to the variable that appears on the left. Thus, the assignment statement

  ```
  total = total + value
  ```

  adds together the current values of the variables `total` and `value` and then stores that sum back in the variable `total`.

- When you assign a new value to a variable, the old value of that variable is lost.

Naming Conventions

- In Python, all names must conform to the syntactic rules for identifiers, which means that the first character must be a letter and the remaining characters must be letters, digits, or the underscore character.
- Beyond these restrictions, Python programmers adopt several conventions to make identifier names easier to recognize:
  - Variable names and function names begin with a lowercase letter and use underscores to separate words, as in the variable name `number_of_students`.
  - Constant names are written entirely in uppercase and use the underscore character to separate words, as in `MAX_HEADROOM`.
  - Class names and function names used as complete programs begin with an uppercase letter and capitalize each internal word, as in `AddTwoNumbers`.

Precedence

- If an expression contains more than one operator, Python uses ***precedence rules*** to determine the evaluation order. The arithmetic operators have the following relative precedence:

`**`	*highest*
unary `-`	
`*` `/` `//` `%`	
`+` `-`	*lowest*

  Thus, Python evaluates the `**` operator first, then the unary `-` operator, then the operators `*`, `/`, `//`, and `%`, and finally the operators `+` and `-`.

- If two operators of the same precedence compete, Python evaluates them from left to right, except for the `**` operator. Parentheses may be used to change the order of operations.

Exercise: Precedence Evaluation

What is the value of the expression at the bottom of the screen?

```
1 * 2 * 3 + (4 + 5) % 6 * (7 + 8) - 9
```

Shorthand Assignments

- Assignment statements often adjust the existing value of a variable by modifying its current value, as in

  ```
  balance = balance + deposit
  ```

- Such statements are so common that Python allows you to shorten this expression to

  ```
  balance += deposit
  ```

- The general form of a ***shorthand assignment*** is

  variable *op*`=` *expression*

  where *op* is any of Python's binary operators. The effect of this statement is the same as

  variable `=` *variable* *op* `(`*expression*`)`

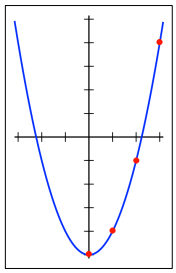
Functions

- A ***function*** is a sequence of statements that has been collected together and given a name, which makes it possible to invoke the complete set of statements simply by supplying the name.
- A function typically takes information from its caller in the form of arguments, perform some computation involving the values of the arguments, and then returns a result to the caller, which continues from the point of the call.
- This notion that functions exist to manipulate information and return results makes functions in programming similar to functions in mathematics, which is the historical reason for the name.

Functions in Mathematics

- The graph at the right shows the values of the function

  $f(x) = x^2 - 5$



- Plugging in a value for x allows you to compute the value of $f(x)$, as follows:

  $f(0) = 0^2 - 5 = -5$
  $f(1) = 1^2 - 5 = -4$
  $f(2) = 2^2 - 5 = -1$
  $f(3) = 3^2 - 5 = 4$

- The Python version of $f(x)$ is

```
def f(x):
    return x**2 - 5
```

Built-In Functions

- To make programming easier, all modern languages include collections of predefined functions. The built-in functions that operate on numeric data appear in the following table:

`abs(`*x*`)`	The absolute value of *x*
`max(`*x*`, `*y*`, ...)`	The largest of the arguments
`min(`*x*`, `*y*`, ...)`	The smallest of the arguments
`round(`*x*`)`	The value of *x* rounded to the nearest integer
`int(`*x*`)`	The value of *x* truncated to an integer
`float(`*x*`)`	The value of *x* converted to floating-point

Importing Library Functions

- In addition to the built-in functions, Python offers a larger set of resources, which are organized into collections called ***libraries***. Before you can use a library function, you must ***import*** that library in one of two ways.
- The most common strategy is to use an `import` statement to acquire access to that library's facilities without specifying any particular functions. For example, the statement

```
import math
```

  indicates that your program will use resources from the `math` library. When you do, you must use the ***fully qualified name***, which includes the library name, as in `math.sqrt`.
- You can also use the `from-import` statement to gain access to specific functions without having to include the library name.

Useful Definitions in the `math` Library

`math.pi`	The mathematical constant π
`math.e`	The mathematical constant *e*
`math.sqrt(`*x*`)`	The square root of *x*
`math.floor(`*x*`)`	Rounds *x* down to the nearest integer
`math.ceil(`*x*`)`	Rounds *x* up to the nearest integer
`math.log(`*x*`)`	The natural logarithm of *x*
`math.log10(`*x*`)`	The common (base 10) logarithm of *x*
`math.exp(`*x*`)`	The inverse logarithm (e^x)
`math.sin(`θ`)`	The sine of θ, measured in radians
`math.cos(`θ`)`	The cosine of θ, measured in radians
`math.atan(`*x*`)`	The principal arctangent of *x*
`math.atan2(`*y*`, `*x*`)`	The arctangent of *y* / *x*
`math.degrees(`θ`)`	Converts from radians to degrees
`math.radians(`θ`)`	Converts from degrees to radians

Writing Your Own Functions

- The general form of a function definition is

```
def name(parameter list):
    statements in the function body
```

  where *name* is the name of the function, and *parameter list* is a list of variables used to hold the values of each argument.
- You can return a value from a function by including a `return` statement, which is usually written as

```
return expression
```

  where *expression* is an expression that specifies the value you want to return.

Examples of Simple Functions

- The following function converts Fahrenheit temperatures to their Celsius equivalent:

```
def ftoc(f):
    return 5 / 9 * (f - 32)
```

- The following function computes the volume of a cylinder of radius `r` and height `h`.

```
def cylinder_volume(r, h):
    return math.pi * r**2 * h
```

Nonnumeric Data

- The arithmetic expressions in the early sections of Chapter 1 enable you to perform numeric computation. Much of the excitement of modern computing, however, lies in the ability of computers to work with other types of data, such as characters, images, sounds, and video.
- As you will learn in Chapter 6, all of these data types are represented inside the machine as sequences of binary digits, or ***bits***. When you are getting started with programming, it is more important to think about data in a more abstract way in which you focus on the conceptual values rather than the underlying representation.

The String Type

- One of the most important data types in any programming language is the ***string type***.
- The domain of the string type is all sequences of characters. In Python, you create a string simply by including that sequence of characters inside quotation marks, as in `"Eric"`.
- The set of operations that can be applied to strings is large, but you don't need to know the entire set. In fact, for the first five chapters in the text, the only string operation you need to know is concatenation, as described on the next slide. You will learn about other operations in Chapter 6.
- All values—including numbers, strings, graphical objects, and values of many other types—can be assigned to variables, passed as arguments to functions, and returned as results.

Concatenation

- One of the most useful operations available for strings is ***concatenation***, which consists of combining two strings end to end with no intervening characters.
- Concatenation is built into Python using the `+` operator. For example, the expression `"ABC" + "DEF"` returns the string `"ABCDEF"`.
- If you use `+` with numeric operands, it signifies addition. If the operands are strings, Python interprets `+` as concatenation.
- To concatenate strings and other types, you need to use the built-in function str to convert values to strings before joining them together, as in

```
"Catch" + str(-22) ⇨ "Catch-22"
```

Built-In Functions for Strings

- Python includes the following built-in functions that operate on string values:

`len(`*s*`)`	The number of characters in *s*
`str(`*x*`)`	The value of *x* converted to a string
`int(`*s*`)`	The string *s* interpreted as an integer
`float(`*s*`)`	The string *s* interpreted as floating-point
`print(`. . .`)`	Prints the arguments separated by spaces
`input(`*prompt*`)`	Reads a string from the user

Running Command-Line Programs

- Although it is possible to import programs into IDLE and run them there, most Python programs are created as text files and then invoked from the command line.
- Python programs specify what happens when the program is run from the command line using the following lines at the end of the program file:

```
if __name__ == "main":
    function()
```

  where *function* is the name of the function that runs the main program.

- Patterned lines of this sort are often called ***boilerplate***. In general, it is more important to memorize boilerplate than to understand it.

The `AddTwoIntegers` Program

```
# File: AddTwoIntegers.py

"""
This program adds two integers entered by the user.
"""

def AddTwoIntegers():
    print("This program adds two integers.")
    n1 = int(input("Enter n1? "))
    n2 = int(input("Enter n2? "))
    sum = n1 + n2
    print("The sum is", sum)

# Startup code

if __name__ == "__main__":
    AddTwoIntegers()
```