Going from Python to C

Darin Brezeale

December 8, 2011

Python is a high-level, interpreted language. C has many of the same types of programming constructs as in Python: arrays, loops, conditionals, functions, file I/O, and so forth.

C is lower level than Python, which means that the programmer must do more of the work (this is the trade-off for having more power). Any task that can be accomplished in Python can be accomplished in C, but it requires more effort and knowledge on the part of the programmer.

For this document, the version of Python is 2.6. The version of C that is discussed is the 1989 version, known as C89. Note that much of the C code is only a piece of a program instead of a complete, working program.

1 Variable Types and Statements

In Python, whenever we wish to store information we just create a variable at the time we need it. We leave it up to the Python interpreter to determine what type of information is being stored: an integer, a floating-pointer number, a string, and so forth.

One of the biggest differences between C and Python is that in C variables must be declared prior to use. When declaring variables, the type of variable (e.g., integer, floating-point, etc) must be known. Statements are terminated with semicolons.
In C, variables must be declared at the top of a block. For the most part, what this means is that at the top of functions all of the variables that will be used in the function must be declared prior to use (except initialization).

Example 1

Python version

```python
a = 10
b = 3.78

total = a / b
```

C version

```c
int a = 10;
double b = 3.78;
double total;

total = a / b;
```

There are times in which we have a variable of one type and as part of a calculation need to treat it as another type. In Python we have functions for temporarily converting a variable. In C this is called **casting** and is done by placing the name of the new type in parentheses.

Example 2

Python version

```
a = 10
b = 4

c = float(a) / b
print c
```

C version

```
int a = 10;
int b = 4;
double c;

c = (double) a / b;
printf("%f", c);
```

2 Basic C Program and Comments

The Python code that we showed above represents a complete program. The C code, however, was not. All code in C must be in a function. The first function to execute will be the `main()` function, which has this form:

```
int main(void)
{
    # code goes here
}
```

Using the code from above, this would look like this:

```
int main(void)
{
    int a = 10;
    double b = 3.78;
    double total;

    total = a / b;
}
```

Just as in Python, in C we have single-line and multi-line comments.

Example 3

Python version

```
# this is a single-line comment
```

or

```
"""
    this is a
    multi-line comment
"""
```

C version

```
// single-line comment
```

or

```
/*
    this is a
    multi-line comment
*/
```

3 Input and Output

Printing in C is typically done via the `printf()` function. C doesn't have built-in functions, so we must include libraries using the `#include` statement at the top of our programs. Printing is not part of the core C language, so the library containing the `printf()` function must be included, much as modules in Python must be imported. The library containing functions for input and output is `stdio.h`.

While strings can be printed in C without format specifiers, numbers must use format specifiers.

Example 4

Python version

```python
a = 5
z = 9.345

print "a=", a, ", z=", z
# or
print "a=%d, z=%.3f\n" % (a, z)
```

C version

```c
#include <stdio.h>

int main(void)
{
  int a = 5;
  double z = 9.345;

  printf("a=%d, z=%.3f\n", a, z);
}
```

In Python, we can both prompt the user for information and get this information using either `raw_input()` (if we want the input to be treated as a string) or `input()` if we want this information to be treated as a number. In C, displaying information to the user and getting information from the user requires two different functions. Displaying information can be done using `printf()` as described above. To get the user's response, we can use `scanf()`.

Example 5

Python version

```python
x = input("Enter an integer:  ")

print "you entered %d" % x
```

C version

```c
#include <stdio.h>

int main(void)
{
  int x;

  printf("Enter an integer:  ");
  scanf("%d", &x);

  printf("you entered %d\n", x);
}
```

The format of `scanf()` is similar to that of `printf()`: we have a string with format specifiers followed by one or more references to existing variables. One big difference, however, is that when the variable for storing the input is a numeric type (e.g., `int`, `double`, etc.), we must precede the variable with an ampersand, known as the **address operator**.

For the most part, the format specifiers used in `scanf()` are the same as with `printf()`: `%d` for variables of type `int`, `%s` for strings, and so forth. A notable exception is that when the variable to store the input in is a `float`, the format specifier is `%f` while the format specifier for storing the input in a `double` is `%lf` (that's a lowercase L). When printing either a `float` or `double`, `%f` can be used.

4 Operators

Operators in C are largely the same as in Python, but there are exceptions and some of the operators that do exist in both don't necessarily work the same way.
In Python, we can perform multiple assignments with one equal sign; in C, each assignment requires its own equal sign.

Example 6

Python version

```python
a, b, c = 1, 2, 3
```

C version

```c
int a = 1;
int b = 2;
int c = 3;
```

or

```c
int a = 1, b = 2, c = 3;
```

We can use this capability in Python to swap the values of two variables without explicitly creating a temporary variable:

Example 7

Python version

```python
a = 10
b = 99

print "a = %d, b = %d" % (a, b)

a, b = b, a

print "a = %d, b = %d" % (a, b)
```

produces

```
a = 10, b = 99
a = 99, b = 10
```

C version

```c
int a = 10, b = 99;
int temp;

printf("a = %d, b = %d\n", a, b);

temp = a;
a = b;
b = temp;

printf("a = %d, b = %d\n", a, b);
```

produces

```
a = 10, b = 99
a = 99, b = 10
```

Python 2.x uses integer division (floor function) while C uses integer division (truncate result). In Python 3.x, the division operator / will use floating point division, so the expression 10/4 will have a value of 2.5.
In both Python and C, when adding, subtracting, dividing, or multiplying a variable by a second variable and reassigning to the first variable, we have multiple ways to do it:

Example 8

Python version

```
x = 10

x = x + 5
x += 6

print x
```

C version

```
int x = 10;

x = x + 5;
x += 6;

printf("%d\n", x);
```

In C we also have special operators for incrementing or decrementing by 1:

```
i++;  /* increment by 1 */
x--;  /* decrement by 1 */
```

There is no exponentiation operator in C, but there is a function in `math.h`. For example, in Python we would use

```
a = 2.6**3
```

to raise 2.6 to the third power. In C, we would use

```
double a;
a = pow(2.6, 3);
```

5 Conditionals

Python has the `if` statement for making decisions. C has the `if` statement, the `switch` statement, and the ternary operator.

Example: Let's say that when `x > 5`, we wish to assign `a` a value of 10 and `b` a value of 76.8.

Example 9

Python version

```
if x > 5:
    a = 10
    b = 76.8
```

C version

```
if (x > 5)
{
    a = 10;
    b = 76.8;
}
```

Note the following differences between the Python and C versions:

- The C version requires parentheses around the test condition and does not terminate the line with a colon.
- While Python requires indentation to show blocks of code, C uses pairs of curly braces. The indentation in the C version is to make the code easier to read.
- Had the C version only had a single assignment statement following the `if` statement, then the curly braces would have been optional. See Example 10.

Example 10

Python version

```python
if x > 5:
    z = 55
```

C version

```c
if (x > 5)
    z = 55;
```

Once again, in the C version the indentation of the assignment statement was not required but is intended to make the code easier to read.

In Python, we can choose among a variety of options using a series of `if` and `elif` statements. We can do the same in C using `else` and `if` statements (there is no `elif` statement in C):

Example 11

Python version

```python
x = 2

if x == 1:
    a = 18
    print ”a is”, a
elif x == 2:
    b = 20
    c = 38
    d = 99
    print ”b is”, b
    print ”c is”, c
    print ”d is”, d
elif x == 3:
    c = 38
    d = 99
    print ”c is”, c
    print ”d is”, d
elif x == 4:
    d = 50
    print ”d is”, d
```

C version

```c
    int x = 2;
    int a, b, c, d;

    if (x == 1)
    {
        a = 18;
        printf(”a is”, a);
    }
    else if (x == 2)
    {
        b = 20;
        c = 38;
        d = 99;
        printf(”b is %d\n”, b);
        printf(”c is %d\n”, c);
        printf(”d is %d\n”, d);
    }
    else if (x == 3)
    {
        c = 38;
        d = 99;
        printf(”c is %d\n”, c);
        printf(”d is %d\n”, d);
    }
    else if (x == 4)
    {
        d = 50;
        printf(”d is”, d);
    }
```

In C, when the choice of what to do is based upon having a specific integer value, then we can use a `switch` statement instead of what is demonstrated in Example 11:

Example 12

```
int x = 2;
int a, b, c, d;

switch( x )
{
case 1: a = 18;
        printf("a is", a);
        break;
case 2: b = 20;
        printf("b is %d\n", b);
case 3: c = 38;
        d = 99;
        printf("c is %d, d is %d\n", c, d);
        break;
case 4: d = 50;
        printf("d is", d);
        break;
}
```

When the `switch` statement is reached, the program jumps to the `case` statement that matches the value of `x`. The `case` statements are simply labels of the next line to execute. Once a certain line has been reached, the program will continue on until all following lines are executed or a `break` statement is reached. A `break` statement causes the flow of the program to move to first line outside of the block associated with the `switch` statement. For the program in Example 8, because `x = 2` the `switch` statement causes the flow of the program to jump to `case 2` where it begins by executing the statement

```
c = 38;
```

The flow continues from here until it reaches the `break` statement at the bottom of `case 3`.

Many times in mathematics, we will apply multiple relational operators at the same time (or at least we think of them as being applied simultaneously). For example, $0 < x < 5$. The reason for my parenthetical comment is that this expression really represents two comparisons. In order for it to be considered true, $0 < x$ must be true and $x < 5$ must be true. Let's see how Python and C deal with this.

operator	Python version	C version	Python example	C example
and	`and`	`&&`	`a and b`	`a && b`
or	`or`	`||`	`a or b`	`a || b`
not	`not`	`!`	`not a`	`!a`

Table 1: logical operators

Example 13

Python version

```python
someInt = 10

if 0 < someInt < 5:
    print "it must be true"
else:
    print "it must be false"
```

produces

```
it must be false
```

C version

```c
int someInt = 10;

if( 0 < someInt < 5 )
    printf("it must be true\n");
else
    printf("it must be false\n");
```

produces

```
it must be true
```

In Python, writing the multiple comparisons like this is legal and works as we expect. It is also legal in C, but doesn't work as we expect. The less than symbols are evaluated left-to-right, which means that `0 < someInt` is evaluated first. Since $0 < 10$, this expression is true and replaced by 1 to indicate it is true. This makes the next comparison $1 < 5$, which is also true. Therefore, the entire thing is considered true in C even though mathematically it isn't. The correct way to do the comparisons in C is to separate them:

```c
    if(0 < someInt && someInt < 5)
```

We have the same logical operators in both Python and C, but their symbols differ; see Table 1.

6 Loops

Python has the `while` loop and the `for` loop (technically, the `for` is an iterator). C has the `for` loop, the `while` loop, and the `do-while` loop.

Example 14

Python version

```python
i = 10
sum = 0

while i < 15 :
    sum = sum + i
    i += 1

print "The sum is", sum
```

C version

```c
int i = 10, sum = 0;

while (i < 15)
{
    sum = sum + i;
    i++;
}

printf("The sum is %d\n", sum);
```

With loops, many of the things that we noted about `if` statements are also true:

- The C version requires parentheses around the test condition and does not terminate the line with a colon.
- C uses pairs of curly braces to enclose a block; the indentation is to make the code easier to read.
- Had the C version only had a single assignment statement following the `while` statement, then the curly braces would have been optional.

Example 15

Python version

```python
sum = 0

for i in range(1, 16):
    sum = sum + i
```

C version

```c
int i;
int sum = 0;

for(i = 1; i < 16; i++)
    sum = sum + i;
```

In Python, the `for` loop is an interator that iterates through the elements of a sequence object. In C, the `for` loop is similar (but not exactly the same) to a `while` loop except that it brings the initialization, test, and change of the counting variable together.

7 Functions

In Python, we can define functions within the body of the main section of the program. In C, we cannot define a new function within another function, including `main()`. Our choices for the location of functions is to define them in the same source-code file as `main()` (above or below it) or in a separate file. If we define the function below `main()` or a separate file, then we must place a **function declaration** above `main()`; see Example 17.

Example 16

Python version

```python
def fx(x, y) :
    z = x + y / 3

    return z


#### main ####
a = 10
b = 3.78

answer = fx( a, b )

print "%.2f\n" % (answer)
```

C version

```c
#include <stdio.h>

double fx(int x, double y)
{
    double z = x + y / 3;

    return z;
}


int main(void)
{
    int a = 10;
    double b = 3.78, answer;

    answer = fx( a, b );

    printf("%.2f\n", answer);
}
```

Example 17 is the same C program as in Example 16, except the function `fx()` is defined below `main()`.

Example 17

```c
#include <stdio.h>

double fx(int x, double y);    # function declaration

int main(void)
{
    int a = 10;
    double b = 3.78, answer;

    answer = fx( a, b );

    printf("%.2f\n", answer);
}

double fx(int x, double y)
{
    double z = x + y / 3;

    return z;
}
```

The two primary differences between function definitions in Python and C are that in C

1. Functions can only return a single object.
2. We must declare the types of the function parameters (i.e., what is being passed to the function) and the type of the object being returned (if any).

If we have a function called `fx()` that receives a single integer and returns a double, then the function declaration would look like this:

```
double fx( int );
```

In this example the parameter type is `int` and the return type is `double`.

8 Arrays

What we called lists in Python are called arrays in C. The two big differences between lists and arrays is that with arrays

1. The elements of an array must be of the same type.
2. We must allocate memory for the values to be stored in the array.

Example 18

Python version

```
data = [10, 20, 30, 40, 50]

i = 0
while i < 5 :
    print "%d" % (data[i])
    i += 1
```

C version

```
#include <stdio.h>

int main(void)
{
  int data[5] = {10,20,30,40,50};
  int i;

  for(i = 0; i < 5; i++)
  {
      printf("%d\n", data[i]);
  }
}
```

In the C version, we allocate enough memory to store five integers by using

```
    int data[5]
```

while also assigning the initial values with the part

```
    = {10, 20, 30, 40, 50};
```

In this case, if we had known that we only wanted to allocate enough memory to store these numbers, we could have used

```
    int data[ ] = {10, 20, 30, 40, 50};
```

Since the quantity is left out of the square brackets, the amount of memory allocated will default to the amount needed to store the numbers initially provided.

There is not a built-in function for determining the number of elements in an array, but we can determine how much memory has been allocated and use this to determine the number of array elements.

Example 19

```
#include <stdio.h>

int main(void)
{
    int data[ ] = {5, 70, 18};
    int size = sizeof( data ) / sizeof( int );

    printf("%d elements\n", size);
}
```

We can use `sizeof` to get the size of an object in bytes. On a computer in which an `int` is 4 bytes, for Example 19 `sizeof( data )` has a value of 12 and `sizeof( int )` has a value of 4, so `size` will be 3.

9 Strings

C doesn't have a true string type. Instead, strings are arrays of characters, where each character is stored as a variable of type `char`.

Example 20

Python version

```
s = "text"

length = len( s )

print "%s has" % s    ,
print "%d characters." % length
```

C version

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char s[ ] = "text";
  int length;

  len = strlen( s );

  printf("%s has ", s);
  printf("%d characters.\n", len);
}
```

The function `strlen()` is in the `string.h` library.

In C, strings must be enclosed in double quotes; single quotes are reserved for single characters.

```
    char letter = 'A';
    char text[] = "cat";
```

In this example, the variable `letter` stores a single character. The array called `text` stores four characters: the three characters forming the word ‘cat’ as well as a character at the end referred to as a terminating null.

Another difference between strings in Python and C is that in C we can't compare strings in the same way we would numbers.

Example 21

Python version

```
a = "cat"
b = "cat"
c = "dog"

if a == b :
    print "%s equals %s" % (a, b)

if a == c :
    print "%s equals %s" % (a, c)
```

C version

```
#include <stdio.h>
#include <string.h>

int main(void)
{
  char a[ ] = "cat";
  char b[ ] = "cat";
  char c[ ] = "dog";

  if (strcmp(a, b) == 0 )
    printf("%s = %s\n", a, b);

  if (strcmp(a, c) == 0 )
    printf("%s = %s\n", a, c);
}
```

The `strcmp()` function returns 0 if two strings are equal.

10 Miscellaneous

In C, there is no Boolean type (at least in C89 or less). We can define constants with the `#define` preprocessor directive that we use like Boolean values. It's not necessary to capitalize the constants, but this is the convention.

Example 22

Python version

```python
status = True

if  status :
    print "It is true."
else :
    print "It is false."
```

C version

```c
#include <stdio.h>

#define TRUE 1
#define FALSE 0

int main(void)
{
    int status = TRUE;

    if (status)
        printf("It is true.\n");
    else
        printf("It is false.\n");

}
```

References

Kernighan, Brian W./Ritchie, Dennis M.: The C Programming Language. 2nd edition. Englewood Cliffs, NJ: Prentice Hall, 1988, ISBN 0131103628

Dennis Ritchie is the creator of the C language and the first edition of this book was THE book on C for many years. This book has a very high value to word count ratio. It is rather terse and not completely up-to-date with the current C standard, but if you want a short introduction to C you might find this book of interest.

King, K.N.: C Programming: A Modern Approach. 2nd edition. New York, NY: W. W. Norton and Company, 2008, ISBN 0393979504

Very good book that includes coverage of C99 and clearly notes the differences between C89 and C99. It is probably best suited for someone with prior programming experience.

Prata, Stephen: C Primer Plus. 5th edition. Sams, 2004

Very good book that includes coverage of C99. Experienced programmers may find it too wordy, but I think that the detailed discussions are good for beginning programmers.