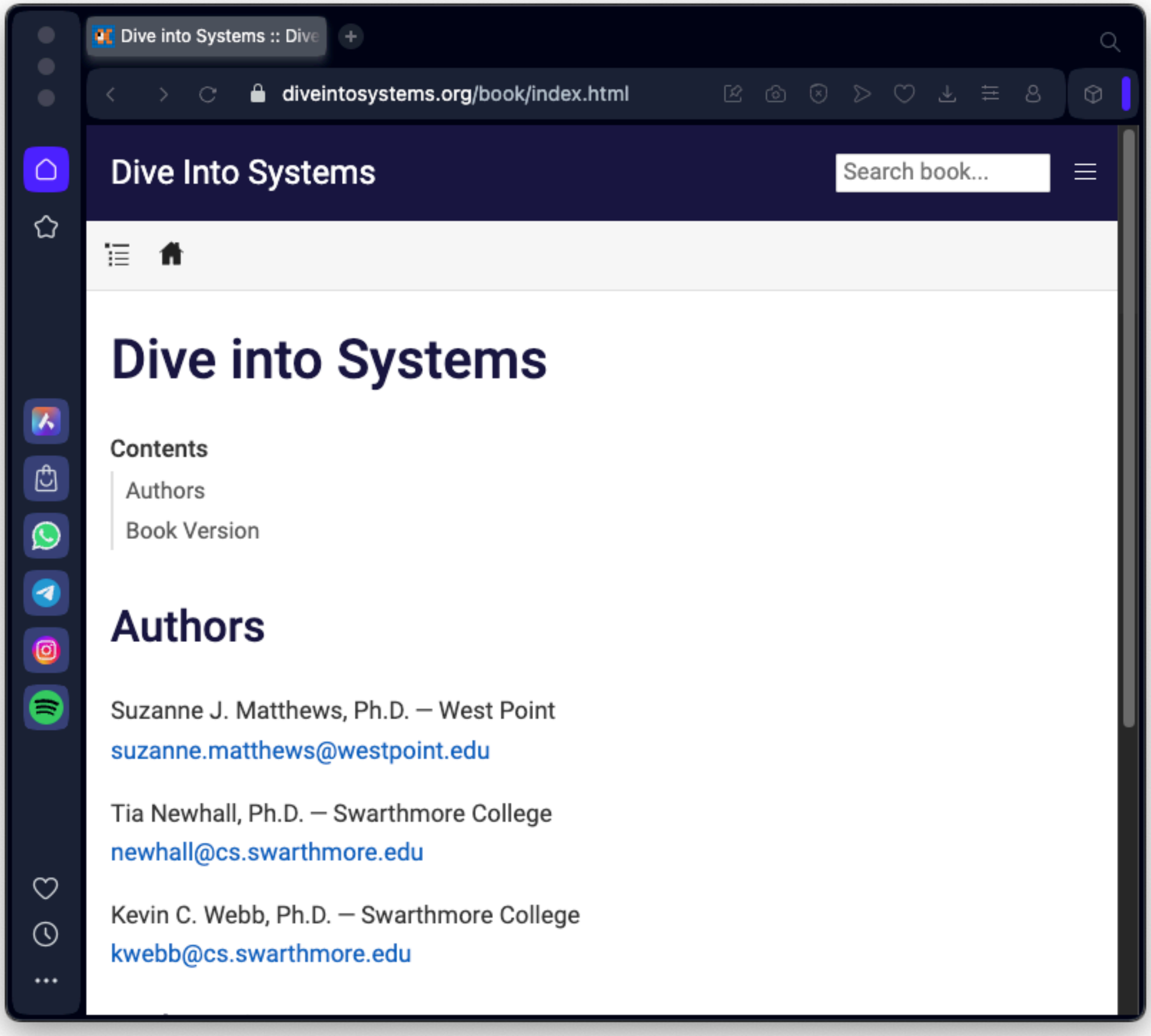
1. By the C, by the C, by the Beautiful C

For COMSC 142

Sam Bowne

Updated Feb 4, 2025

Free online textbook


Dive into Systems :: Dive
diveintosystems.org/book/index.html
Dive Into Systems
Search book...

Dive into Systems

Contents

- Authors
- Book Version

Authors

Suzanne J. Matthews, Ph.D. — West Point
suzanne.matthews@westpoint.edu

Tia Newhall, Ph.D. — Swarthmore College
newhall@cs.swarthmore.edu

Kevin C. Webb, Ph.D. — Swarthmore College
kwebb@cs.swarthmore.edu

- https://diveintosystems.org/book/index.html

Topics

- 1.1. Getting Started Programming in C
- 1.2. Input/Output (printf and scanf)
- 1.3. Conditionals and Loops
- 1.4. Functions
- 1.5. Arrays and Strings
- 1.6. Structs
- 1.7. Summary

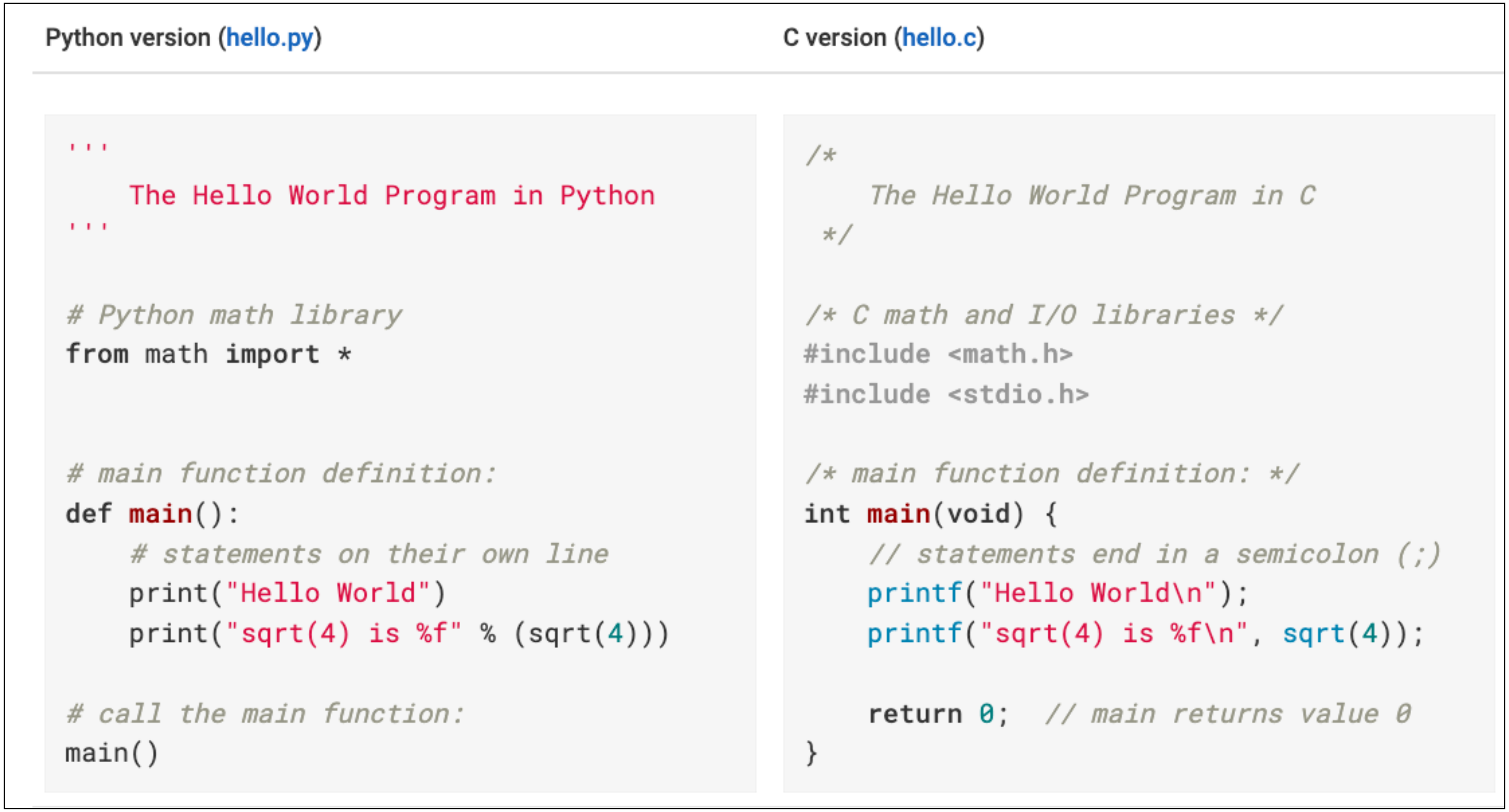
C

- High-level programming language
- Less abstracted from machine language than languages like Python, Java, Ruby, or C++
- No support for
  - Object-oriented programming
  - High-level abstractions like strings, lists, or dictionaries in Python
- C code runs more efficiently
  - The choice where low-level control and efficiency are critical

1.1. Getting Started Programming in C

Hello World

- C requires the main() function



Python version (hello.py)

```python
'''
    The Hello World Program in Python
'''

# Python math library
from math import *


# main function definition:
def main():
    # statements on their own line
    print("Hello World")
    print("sqrt(4) is %f" % (sqrt(4)))

# call the main function:
main()
```

C version (hello.c)

```c
/*
    The Hello World Program in C
 */

/* C math and I/O libraries */
#include <math.h>
#include <stdio.h>

/* main function definition: */
int main(void) {
    // statements end in a semicolon (;)
    printf("Hello World\n");
    printf("sqrt(4) is %f\n", sqrt(4));

    return 0;  // main returns value 0
}
```

Python vs. C

- Comments:
  - In Python, multiline comments begin and end with **'''**, and single-line comments begin with **#.**
  - In C, multiline comments begin with **/*** and end with ***/**, and single-line comments begin with **//**.
- Importing library code:
  - In Python, libraries are included (imported) using **import**.
  - In C, libraries are included (imported) using **#include**. All **#include** statements appear at the top of the program, outside of function bodies.

Python vs. C

- Blocks:
  - In Python, indentation denotes a block.
  - In C, blocks (for example, function, loop, and conditional bodies) start with **{** and end with **}**.
- The main function:
  - In Python, no main() is required
  - In C, **int main(void){ }** defines the main function.
    - The main function returns a value of type int
    - Returns 0 if no error
    - The **void** means it doesn’t expect parameters

Python vs. C

- Statements:
  - In Python, each statement is on a separate line.
  - In C, each statement ends with a semicolon **;**
    - Statements must be within the body of some function (in main in this example).
- Output:
  - In Python, the print function prints a formatted string.
  - In C, the printf function prints a formatted string.
    - **%f** indicates floating point

C

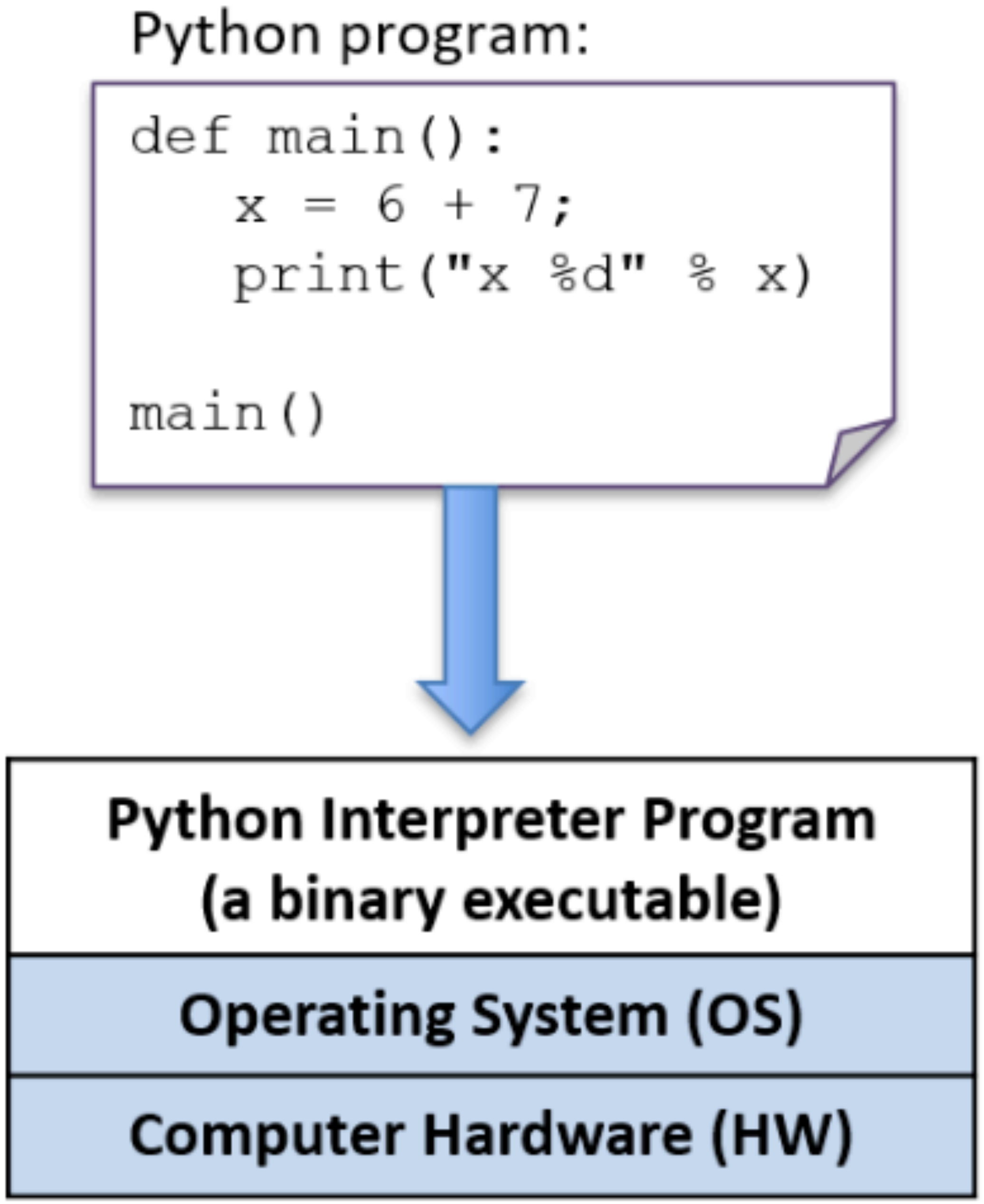
- Indentation:
  - In C, indentation doesn’t have meaning
  - It’s good programming style to indent statements
- Output:
  - C’s printf function doesn’t automatically print a newline character at the end
  - Programmers need to explicitly specify a newline character (**\n**) in the format string

main() Function

- A C program must have a function named **main**, and its return type must be **int**.
- The C main function has an explicit return statement to return an int value (by convention, main should return 0 if the main function is successfully executed without errors).

Python Interpreter

- The interpreter is like a virtual machine Python runs on
- Converts Python statements to machine code at runtime



```
def main():
    x = 6 + 7;
    print("x %d" % x)

main()
```

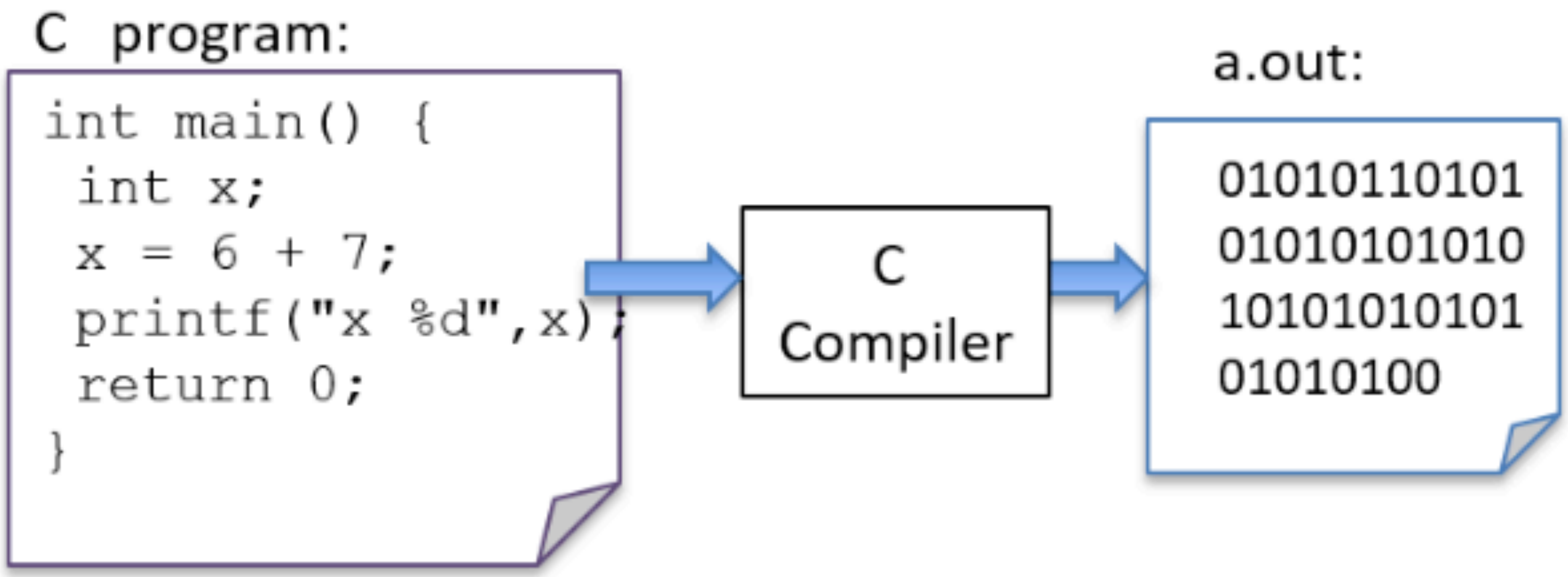
Python: Interpreted Execution

1.1.1. Compiling and Running C Programs

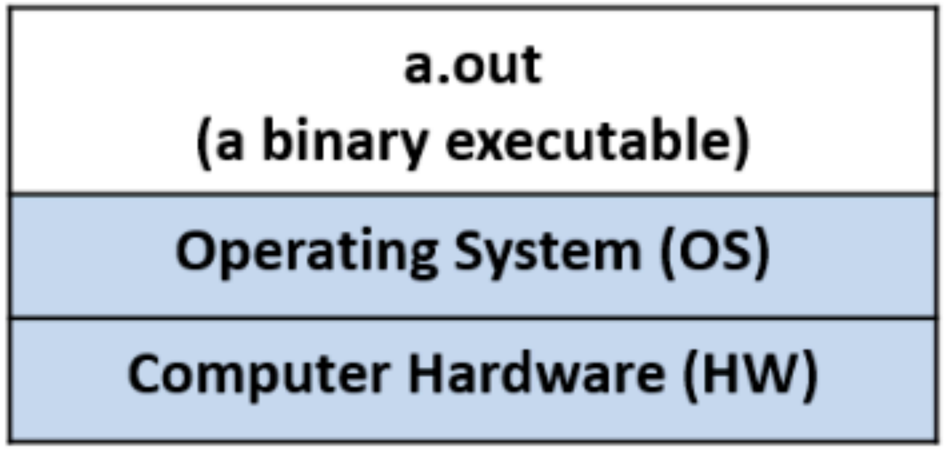
- C code is **compiled** into a binary executable
- Default name **a.out**
- Command lines

$ **gcc hello.c**

$ **./a.out**



```
int main() {
 int x;
 x = 6 + 7;
 printf("x %d",x);
 return 0;
}
```



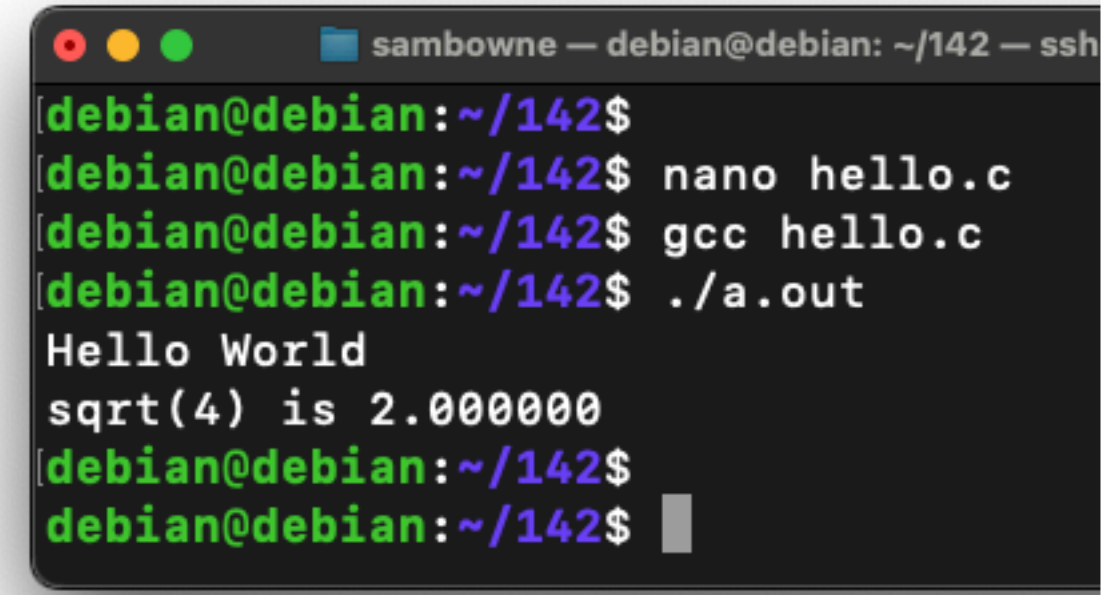
C: First compiled into a.out
Then direct execution of a.out

Demo

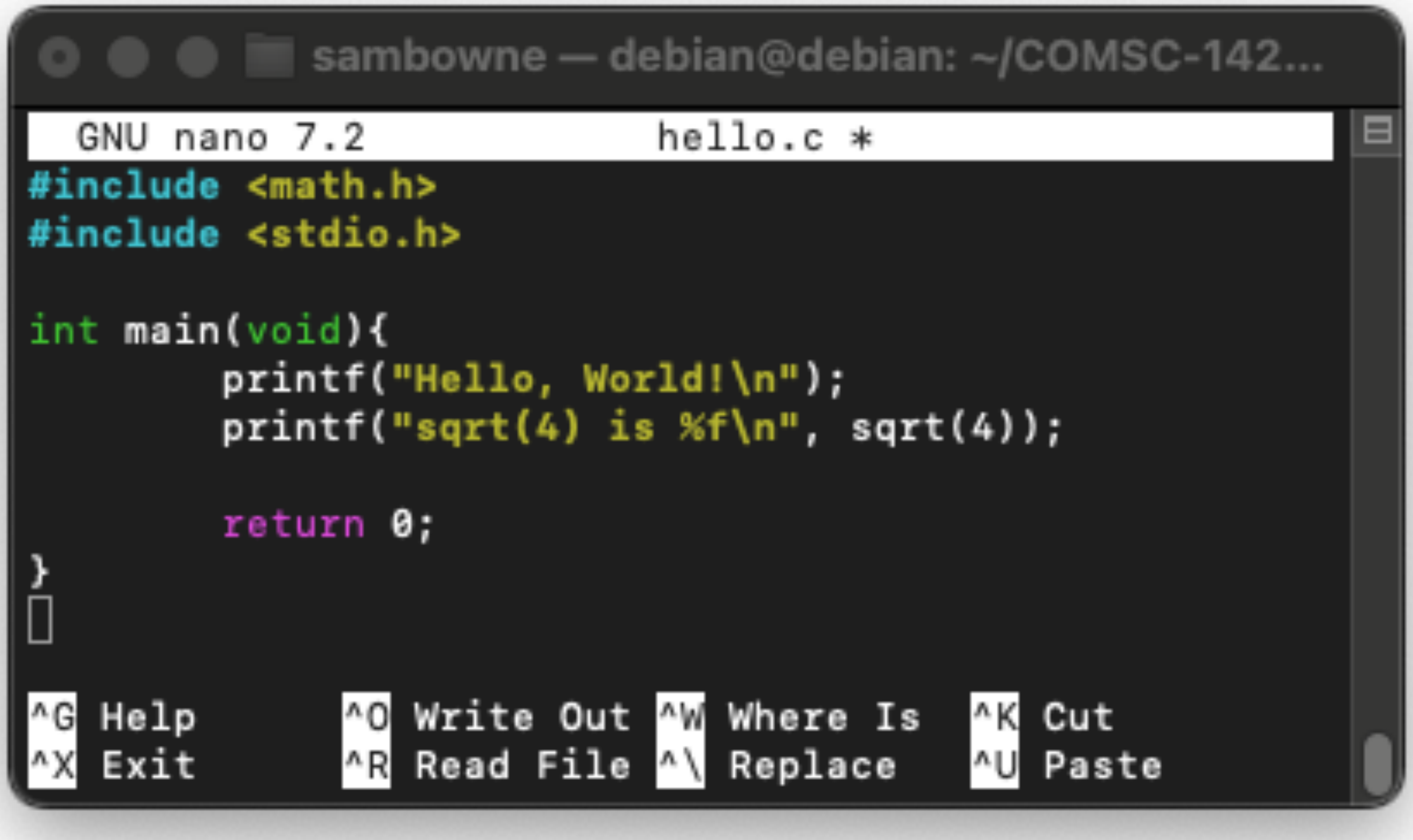
- On Debian, you must first do this to install gcc

```
sudo apt update
sudo apt install build-essential
```

Demo



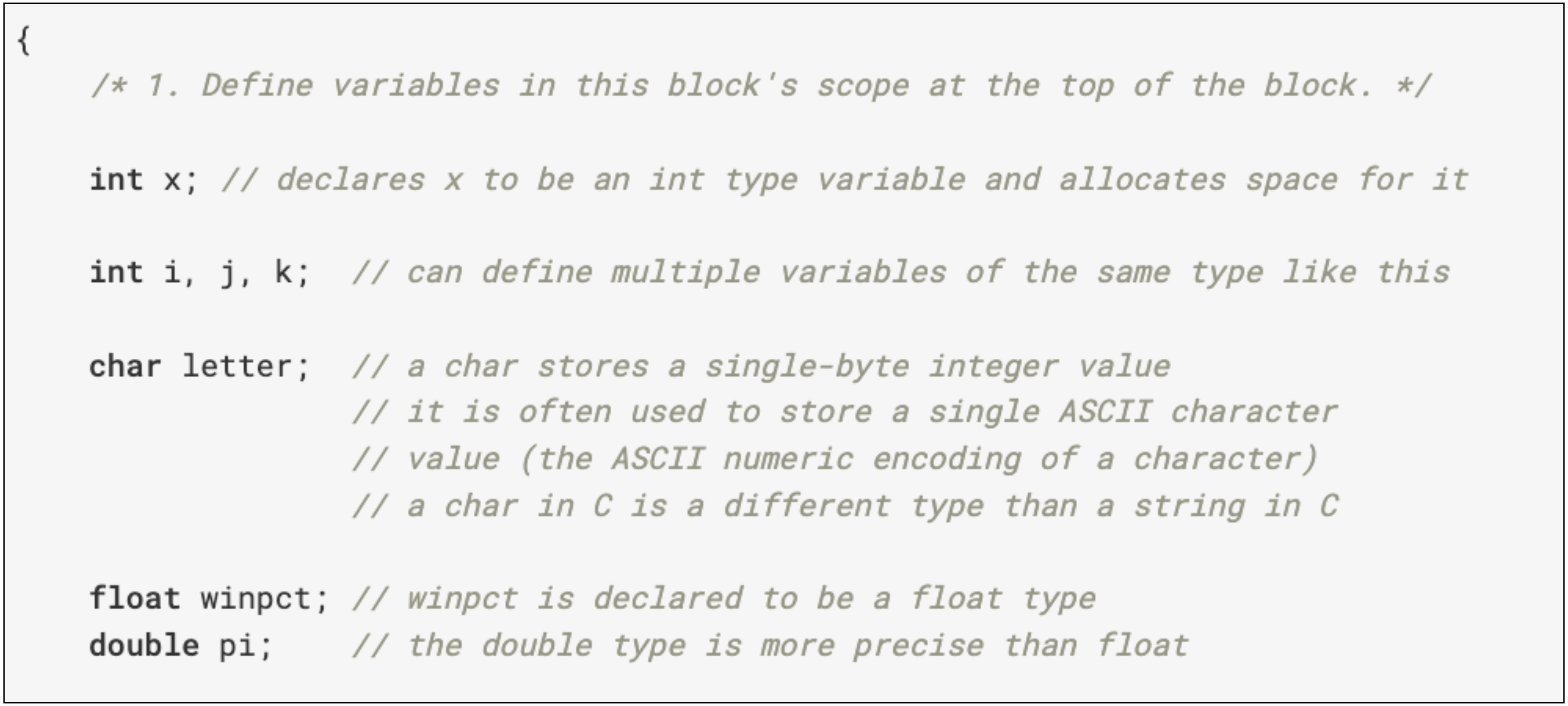
```
debian@debian:~/142$
debian@debian:~/142$ nano hello.c
debian@debian:~/142$ gcc hello.c
debian@debian:~/142$ ./a.out
Hello World
sqrt(4) is 2.000000
debian@debian:~/142$
debian@debian:~/142$ 
```



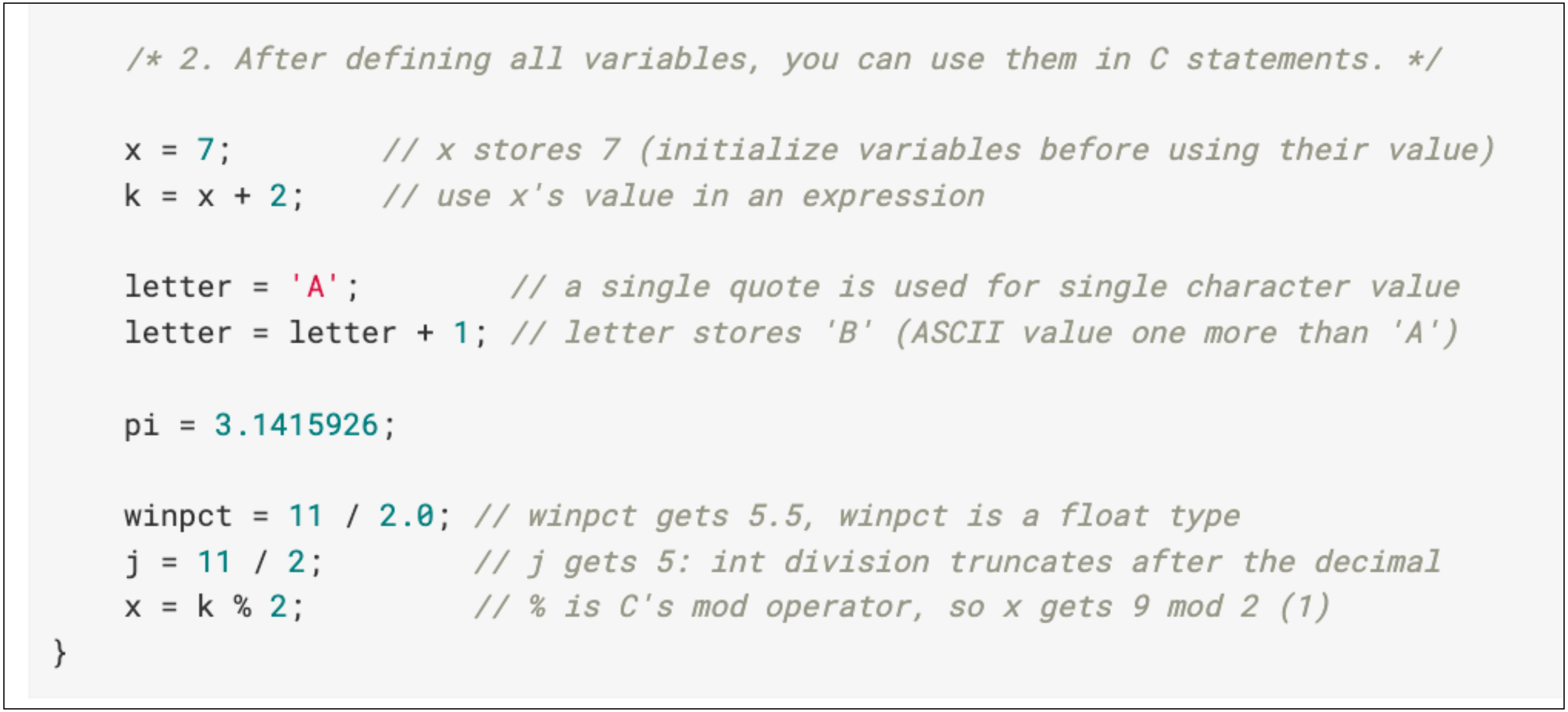
```
  GNU nano 7.2                 hello.c *
#include <math.h>
#include <stdio.h>

int main(void){
        printf("Hello, World!\n");
        printf("sqrt(4) is %f\n", sqrt(4));

        return 0;
}


^G Help       ^O Write Out ^W Where Is  ^K Cut
^X Exit       ^R Read File ^\ Replace   ^U Paste
```

1.1.2. Variables and C Numeric Types

- Variables have **scope** and **type**
- Scope
  - Where in the program it can be used
- Type
  - What range of values it can have

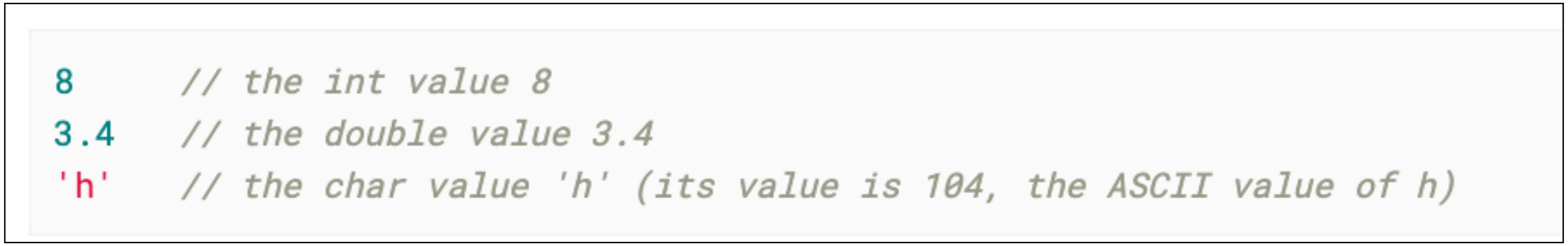


```
{
    /* 1. Define variables in this block's scope at the top of the block. */

    int x; // declares x to be an int type variable and allocates space for it

    int i, j, k;  // can define multiple variables of the same type like this

    char letter;  // a char stores a single-byte integer value
                  // it is often used to store a single ASCII character
                  // value (the ASCII numeric encoding of a character)
                  // a char in C is a different type than a string in C

    float winpct; // winpct is declared to be a float type
    double pi;    // the double type is more precise than float
```



```c
    /* 2. After defining all variables, you can use them in C statements. */

    x = 7;        // x stores 7 (initialize variables before using their value)
    k = x + 2;    // use x's value in an expression

    letter = 'A';        // a single quote is used for single character value
    letter = letter + 1; // letter stores 'B' (ASCII value one more than 'A')

    pi = 3.1415926;

    winpct = 11 / 2.0; // winpct gets 5.5, winpct is a float type
    j = 11 / 2;        // j gets 5: int division truncates after the decimal
    x = k % 2;         // % is C's mod operator, so x gets 9 mod 2 (1)
}
```

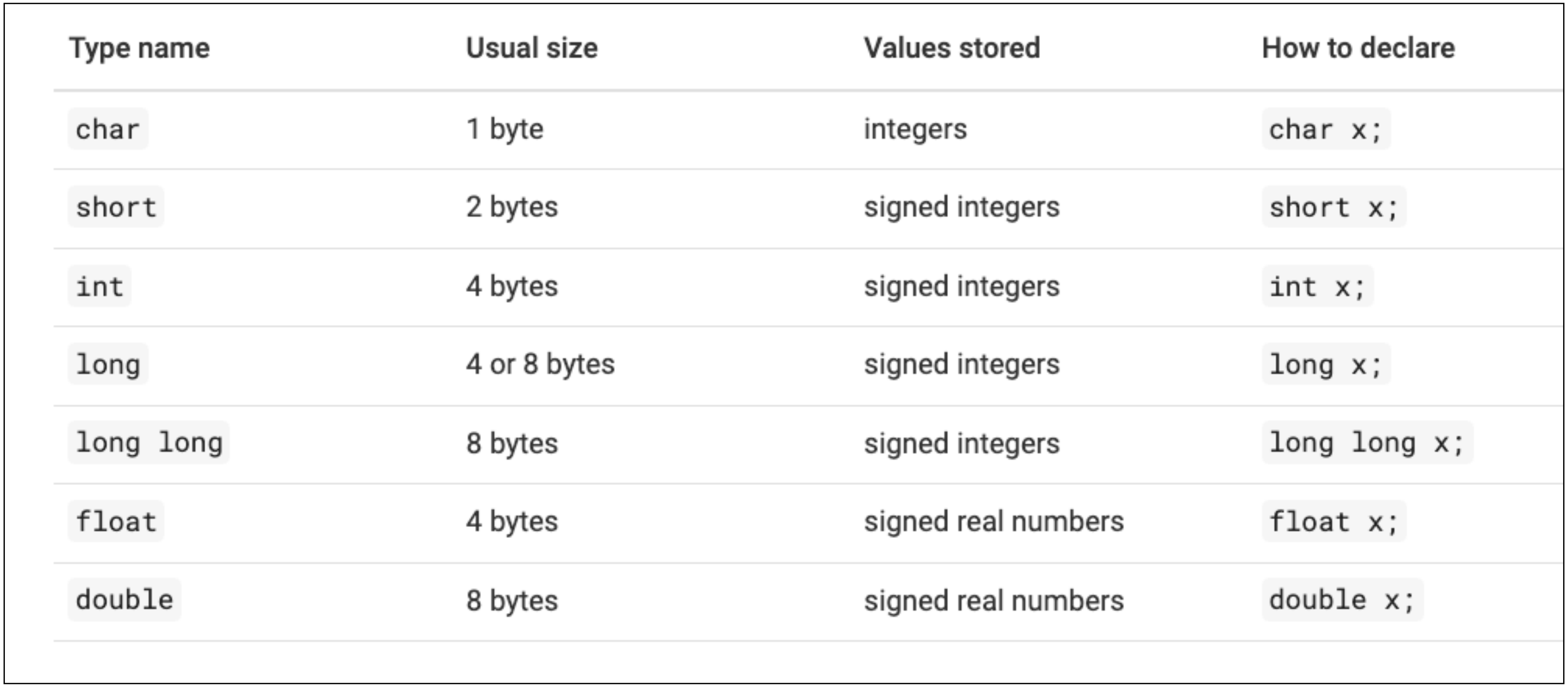
1.1.3. C Types



```
8      // the int value 8
3.4    // the double value 3.4
'h'    // the char value 'h' (its value is 104, the ASCII value of h)
```

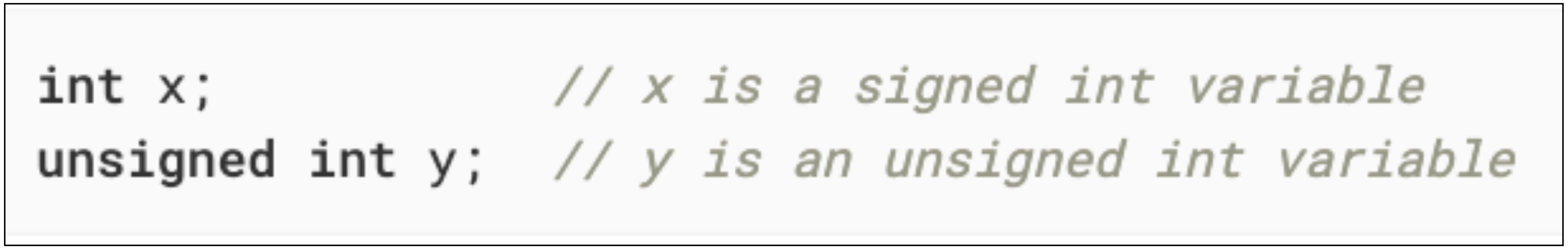
- The C char type stores a numeric value. However, it’s often used by programmers to store the value of an ASCII character.
- The thing called a "string" in C is just a byte array
  - printf("this is a C string\n");

C Numeric Types



Type name	Usual size	Values stored	How to declare
`char`	1 byte	integers	`char x;`
`short`	2 bytes	signed integers	`short x;`
`int`	4 bytes	signed integers	`int x;`
`long`	4 or 8 bytes	signed integers	`long x;`
`long long`	8 bytes	signed integers	`long long x;`
`float`	4 bytes	signed real numbers	`float x;`
`double`	8 bytes	signed real numbers	`double x;`

Unsigned Integers

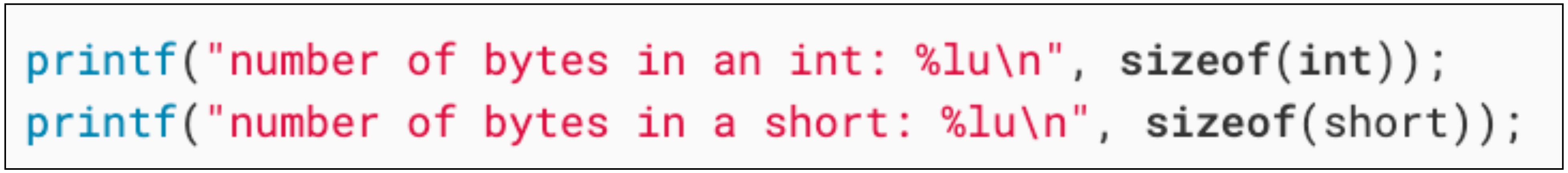


```
int x;               // x is a signed int variable
unsigned int y;      // y is an unsigned int variable
```

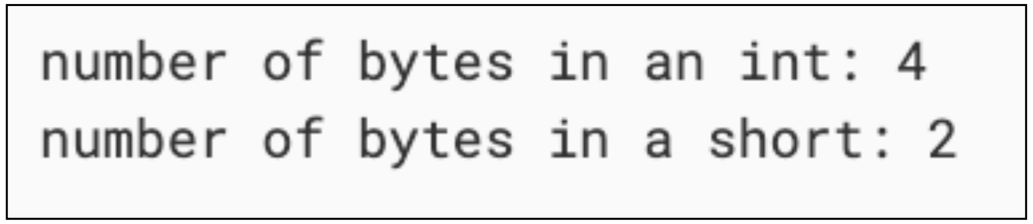
- **char** might be signed or unsigned
    - Depending on the implementation

sizeof

- Shows the actual sizes of types, which may vary



```
printf("number of bytes in an int: %lu\n", sizeof(int));
printf("number of bytes in a short: %lu\n", sizeof(short));
```



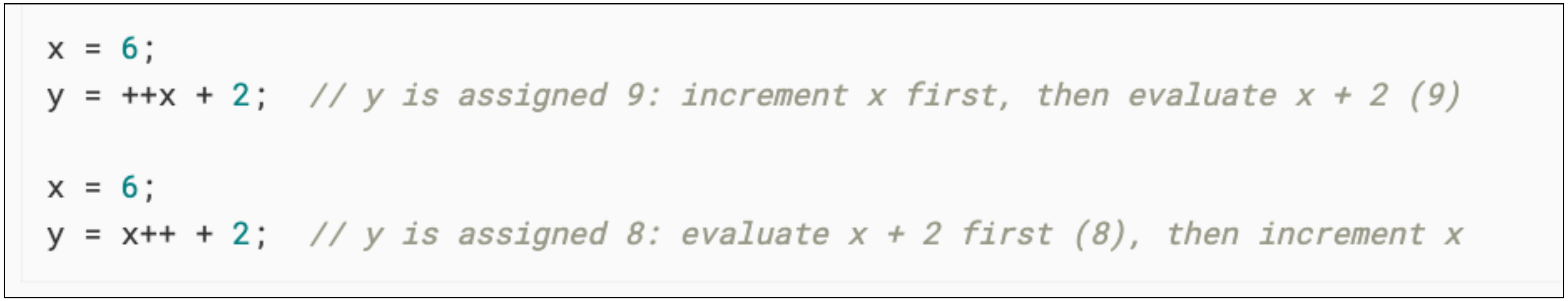
```
number of bytes in an int: 4
number of bytes in a short: 2
```

Arithmetic Operators

- add (+) and subtract (-)
- multiply (*), divide (/), and mod (%)
- assignment (=)
- assignment with update (+=, -=, *=, /=, and %=)
- increment (++) and decrement (--)

Pre- vs. Post-increment

- **++x** increment x first, then use its value.
- **x++** use x’s value first, then increment it.



```
x = 6;
y = ++x + 2;  // y is assigned 9: increment x first, then evaluate x + 2 (9)

x = 6;
y = x++ + 2;  // y is assigned 8: evaluate x + 2 first (8), then increment x
```

Kahoot!

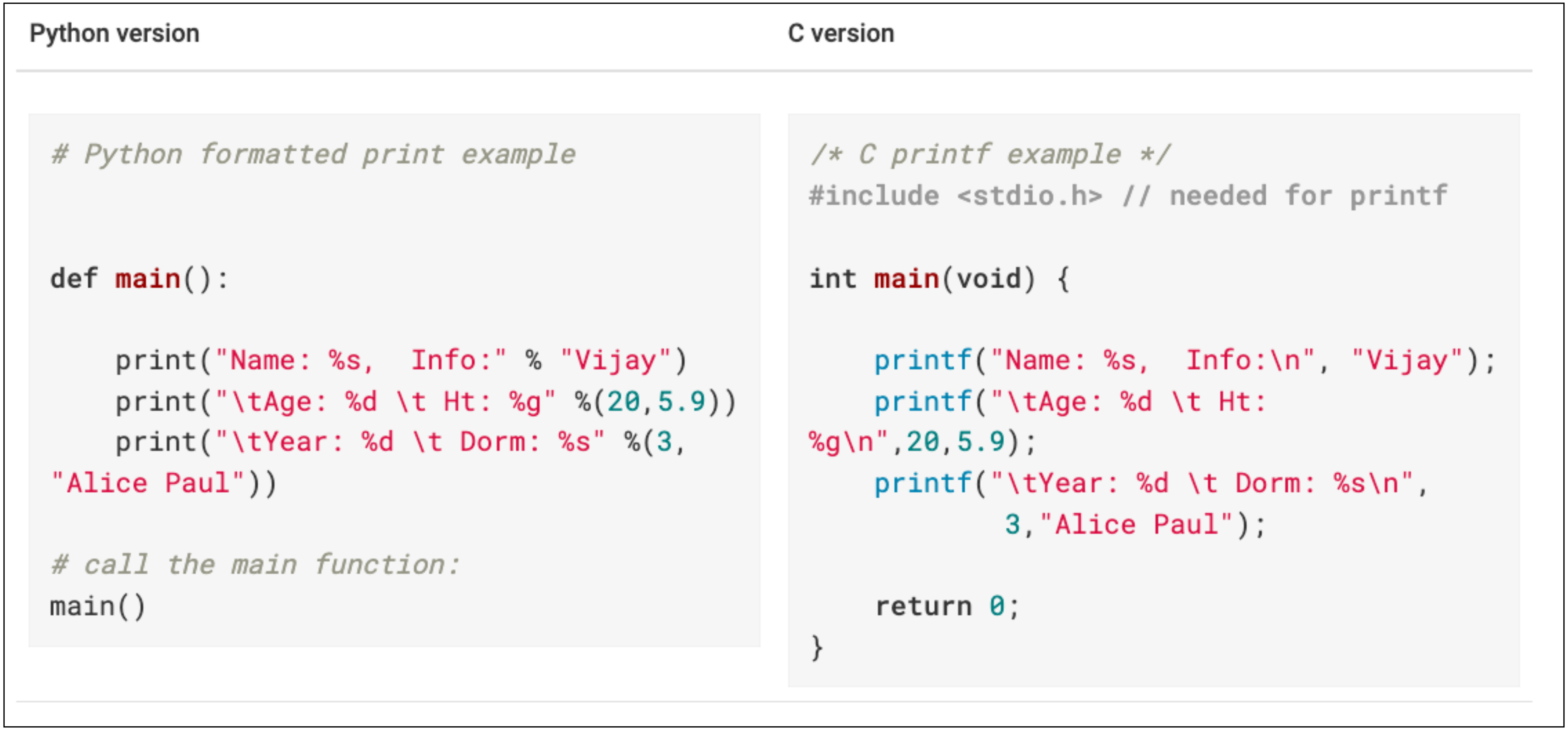
Ch 1a

1.2. Input/Output (printf and scanf)

stdio.h

- To use printf and scanf, you must put this at the top of your .c file
  - **#include <stdio.h>**

1.2.1. printf

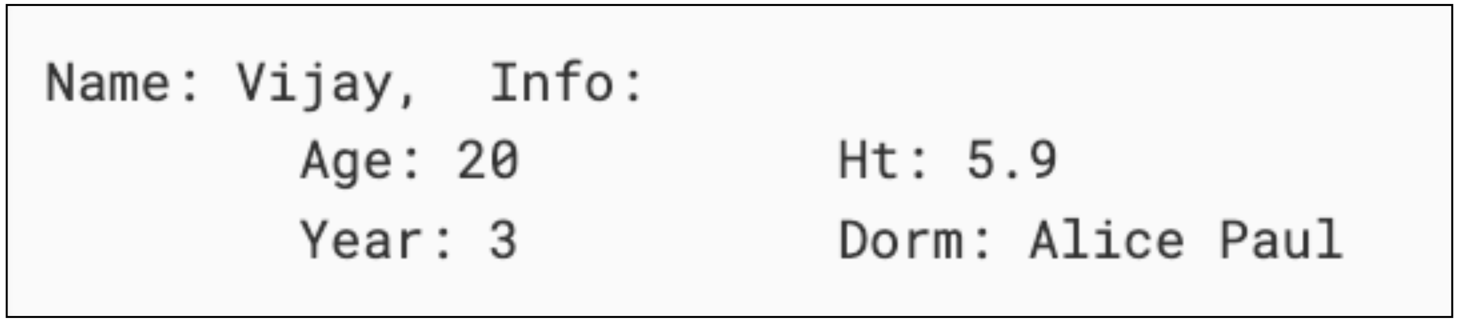


Python version

```python
# Python formatted print example


def main():

    print("Name: %s,  Info:" % "Vijay")
    print("\tAge: %d \t Ht: %g" %(20,5.9))
    print("\tYear: %d \t Dorm: %s" %(3,
"Alice Paul"))

# call the main function:
main()
```

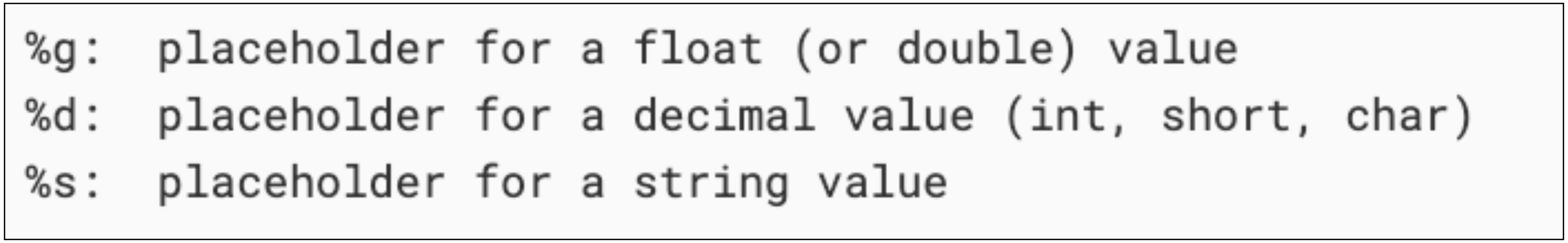
C version

```c
/* C printf example */
#include <stdio.h> // needed for printf

int main(void) {

    printf("Name: %s,  Info:\n", "Vijay");
    printf("\tAge: %d \t Ht:
%g\n",20,5.9);
    printf("\tYear: %d \t Dorm: %s\n",
            3,"Alice Paul");

    return 0;
}
```



```
Name: Vijay,  Info:
        Age: 20          Ht: 5.9
        Year: 3          Dorm: Alice Paul
```

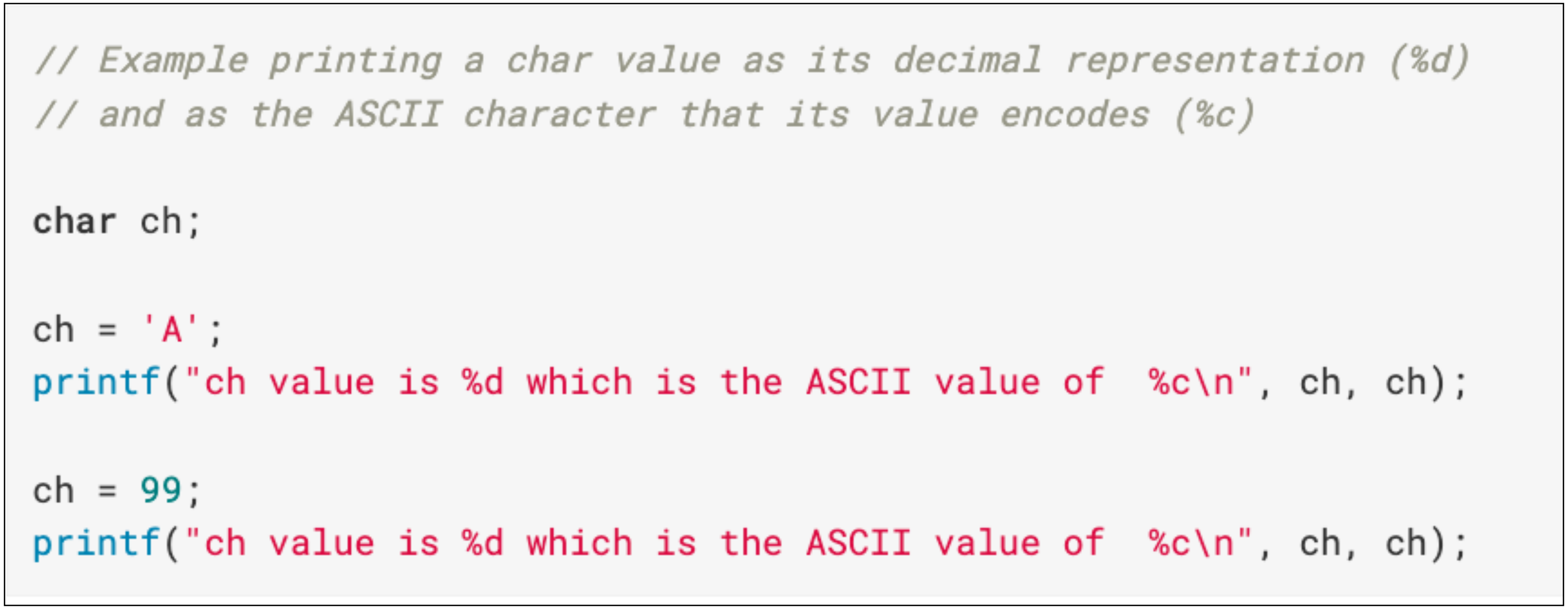
Formatting Placeholders



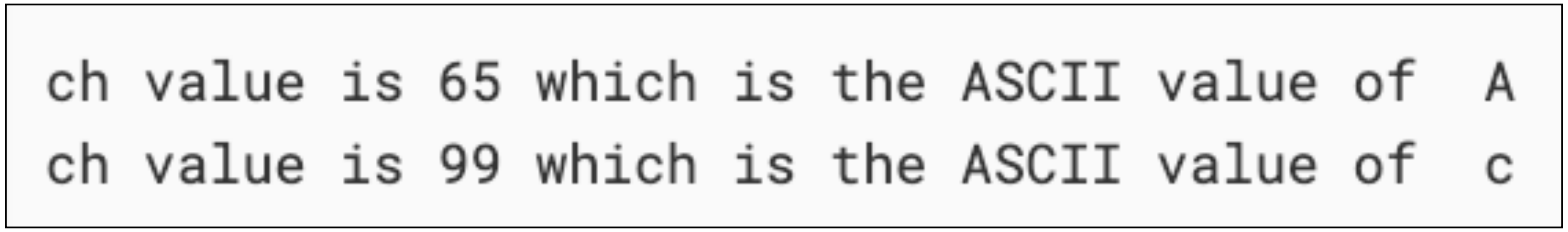
```
%g:  placeholder for a float (or double) value
%d:  placeholder for a decimal value (int, short, char)
%s:  placeholder for a string value
```

- **%c** for a character

char as a number

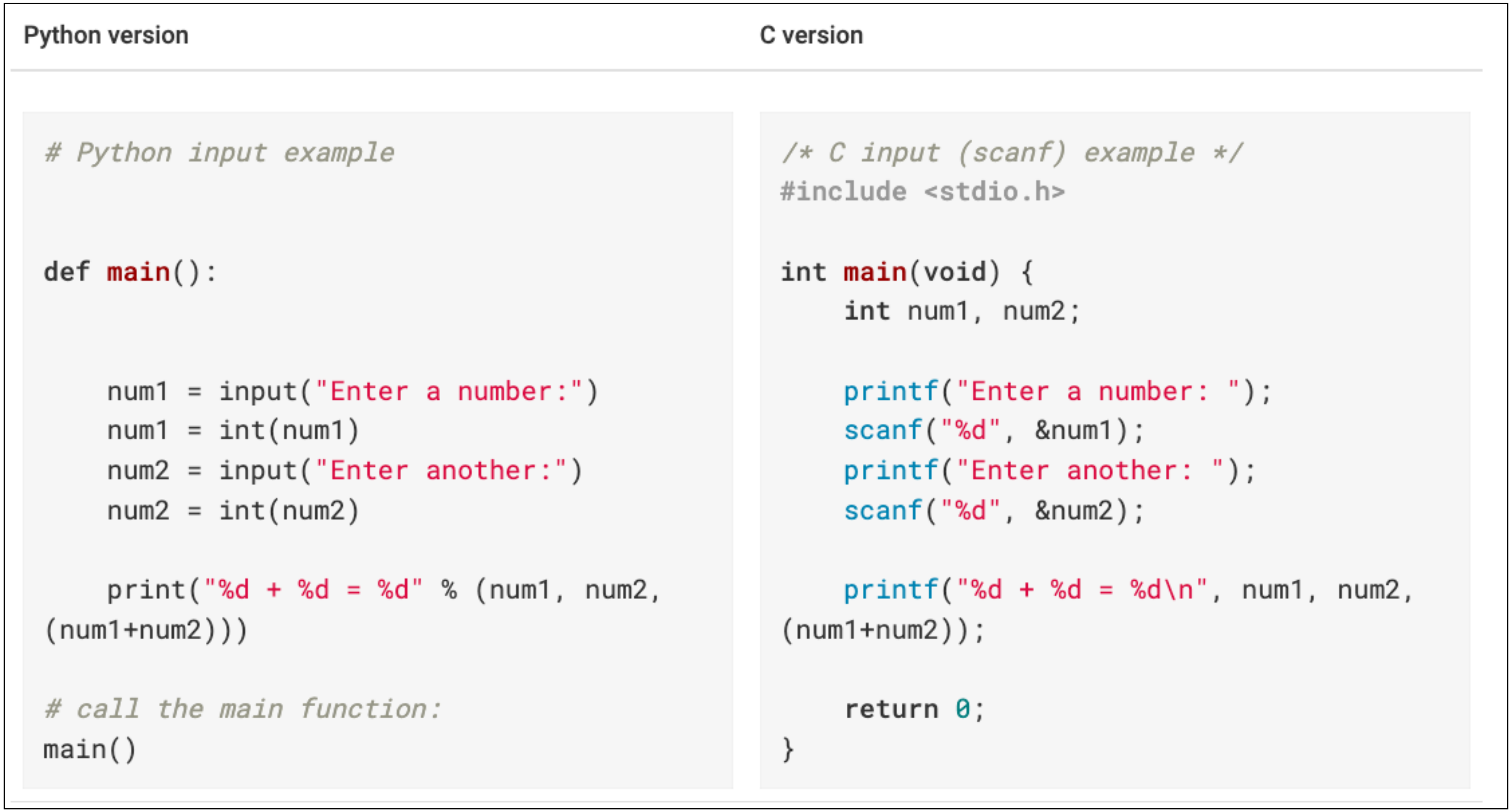


```
// Example printing a char value as its decimal representation (%d)
// and as the ASCII character that its value encodes (%c)

char ch;

ch = 'A';
printf("ch value is %d which is the ASCII value of  %c\n", ch, ch);

ch = 99;
printf("ch value is %d which is the ASCII value of  %c\n", ch, ch);
```



```
ch value is 65 which is the ASCII value of  A
ch value is 99 which is the ASCII value of  c
```

1.2.2. scanf

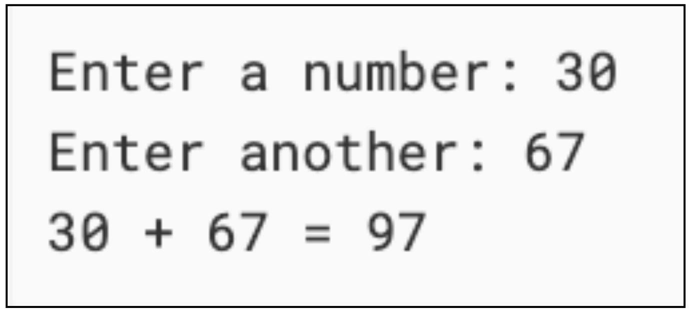


Python version

```python
# Python input example


def main():


    num1 = input("Enter a number:")
    num1 = int(num1)
    num2 = input("Enter another:")
    num2 = int(num2)

    print("%d + %d = %d" % (num1, num2,
(num1+num2)))

# call the main function:
main()
```

C version

```c
/* C input (scanf) example */
#include <stdio.h>

int main(void) {
    int num1, num2;

    printf("Enter a number: ");
    scanf("%d", &num1);
    printf("Enter another: ");
    scanf("%d", &num2);

    printf("%d + %d = %d\n", num1, num2,
(num1+num2));

    return 0;
}
```

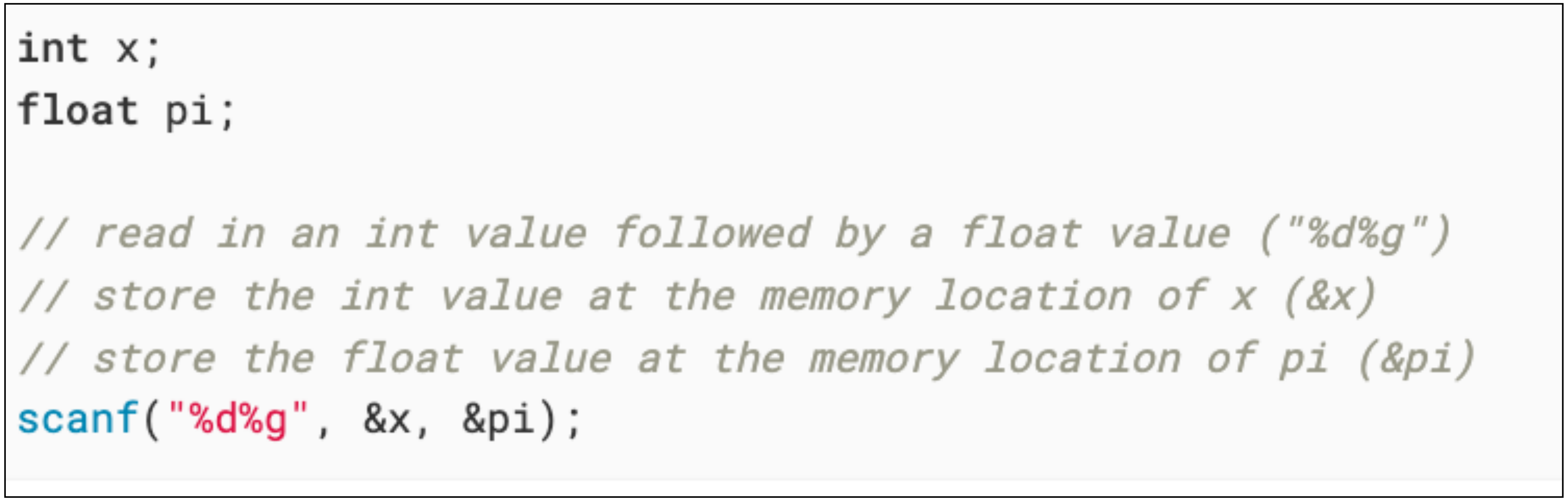


```
Enter a number: 30
Enter another: 67
30 + 67 = 97
```

& Operator

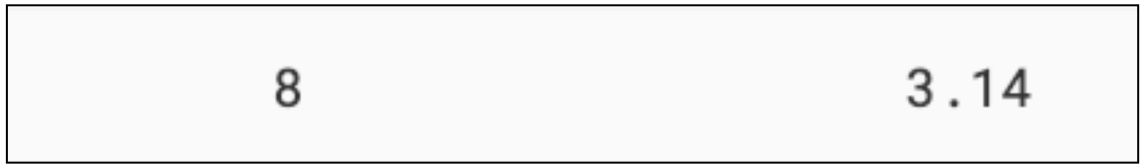
- Prefixing the name of a variable with the **&** operator
- produces the location of that variable in the program’s memory — the memory address of the variable.

Reading Two Values



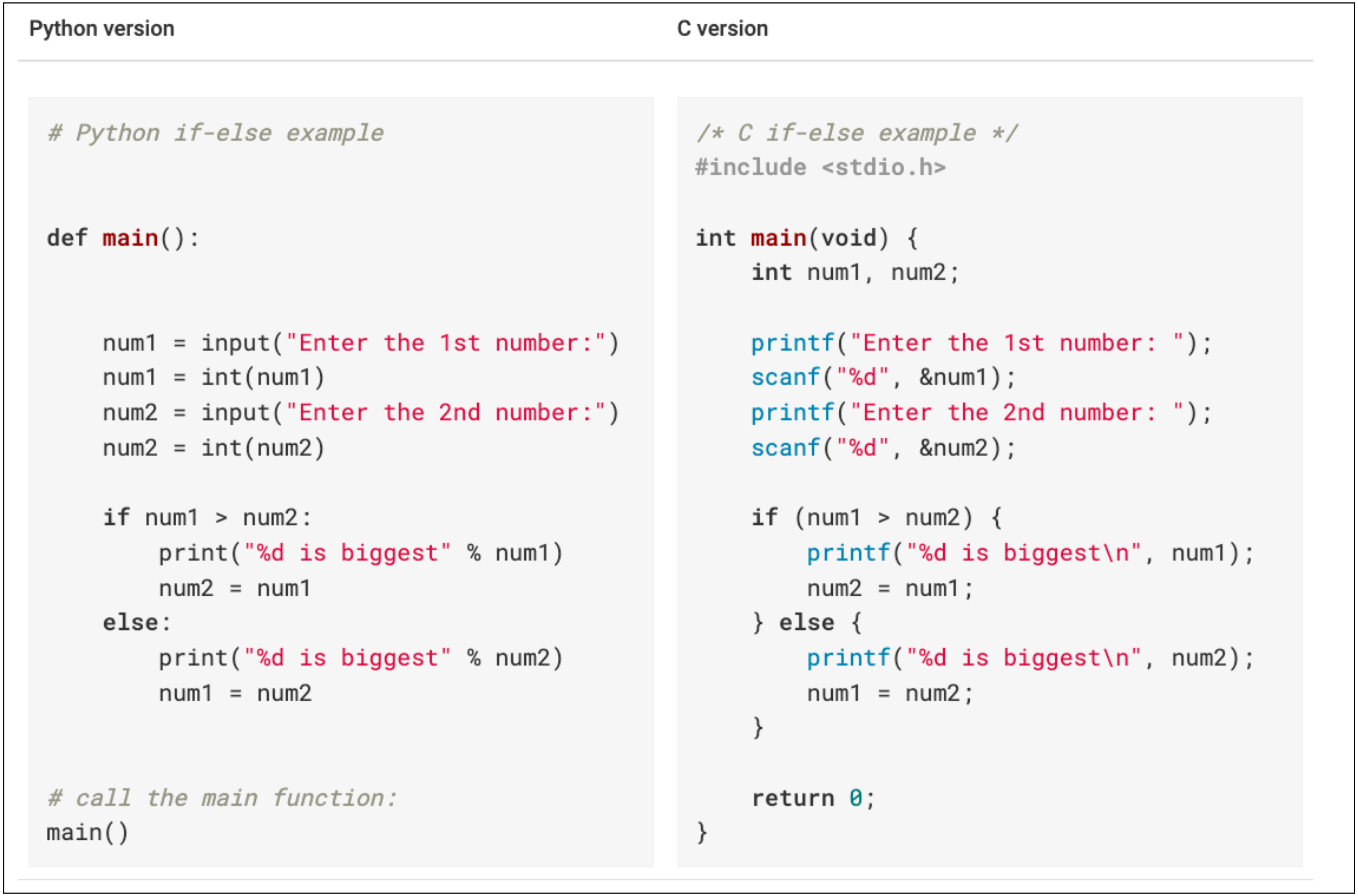
```c
int x;
float pi;

// read in an int value followed by a float value ("%d%g")
// store the int value at the memory location of x (&x)
// store the float value at the memory location of pi (&pi)
scanf("%d%g", &x, &pi);
```

- Input values may be separated by any amount of whitespace
    - spaces, tabs, or newlines



```
        8                   3.14
```

1.3. Conditionals and Loops



Python version

```python
# Python if-else example


def main():


    num1 = input("Enter the 1st number:")
    num1 = int(num1)
    num2 = input("Enter the 2nd number:")
    num2 = int(num2)

    if num1 > num2:
        print("%d is biggest" % num1)
        num2 = num1
    else:
        print("%d is biggest" % num2)
        num1 = num2


# call the main function:
main()
```

C version

```c
/* C if-else example */
#include <stdio.h>

int main(void) {
    int num1, num2;

    printf("Enter the 1st number: ");
    scanf("%d", &num1);
    printf("Enter the 2nd number: ");
    scanf("%d", &num2);

    if (num1 > num2) {
        printf("%d is biggest\n", num1);
        num2 = num1;
    } else {
        printf("%d is biggest\n", num2);
        num1 = num2;
    }

    return 0;
}
```

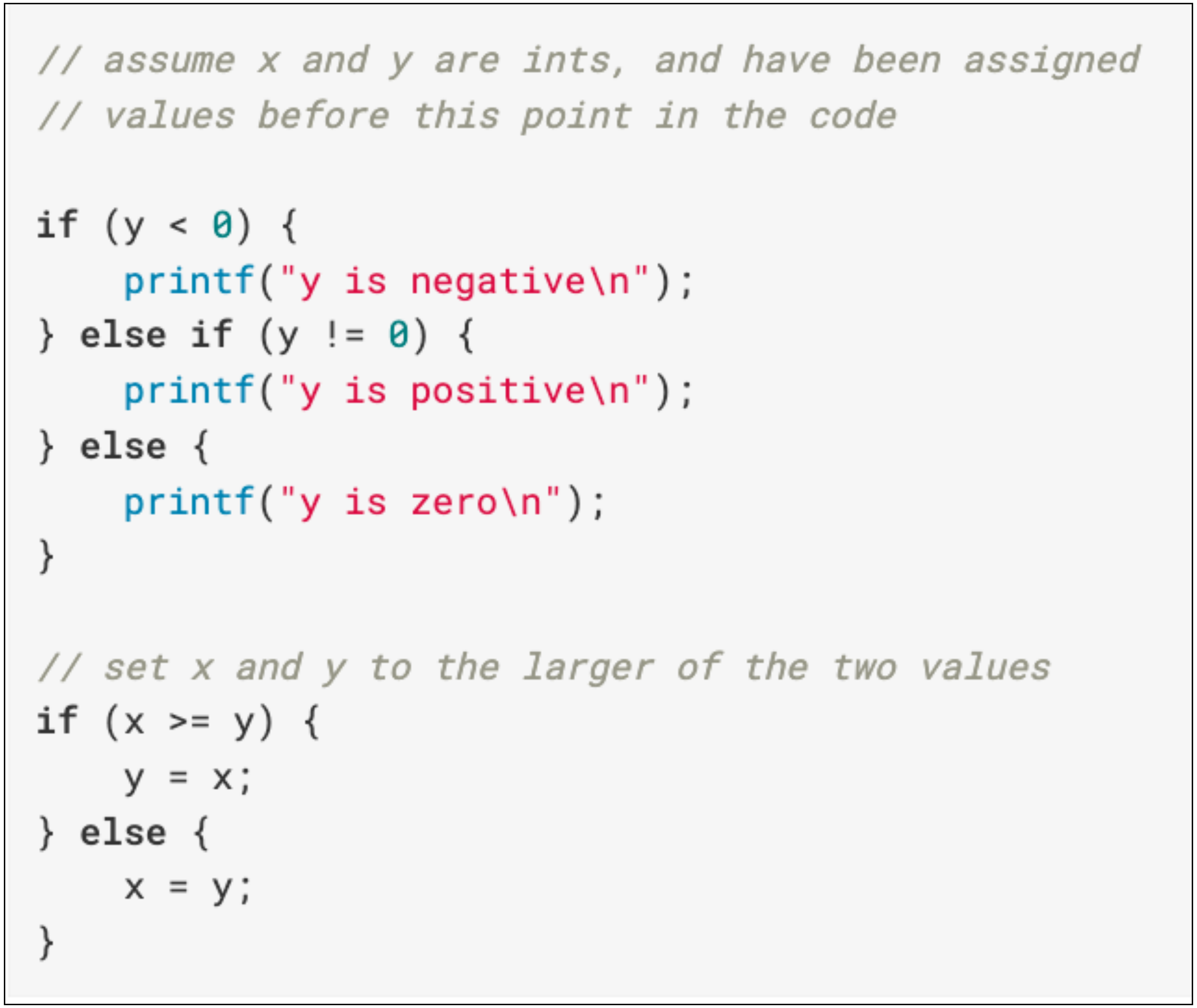
1.3.1. Boolean Values in C

- C doesn’t provide a Boolean type with true or false values.
- Instead, integer values evaluate to **true** or **false** when used in conditional statements.
  - zero (**0**) evaluates to false
  - nonzero (any positive or negative value) evaluates to true

Relational Operators

- equality **==** and inequality **!=**
- comparison operators
  - less than **<**
  - less than or equal **<=**
  - greater than **>**
  - greater than or equal **>=**

Relational Operators



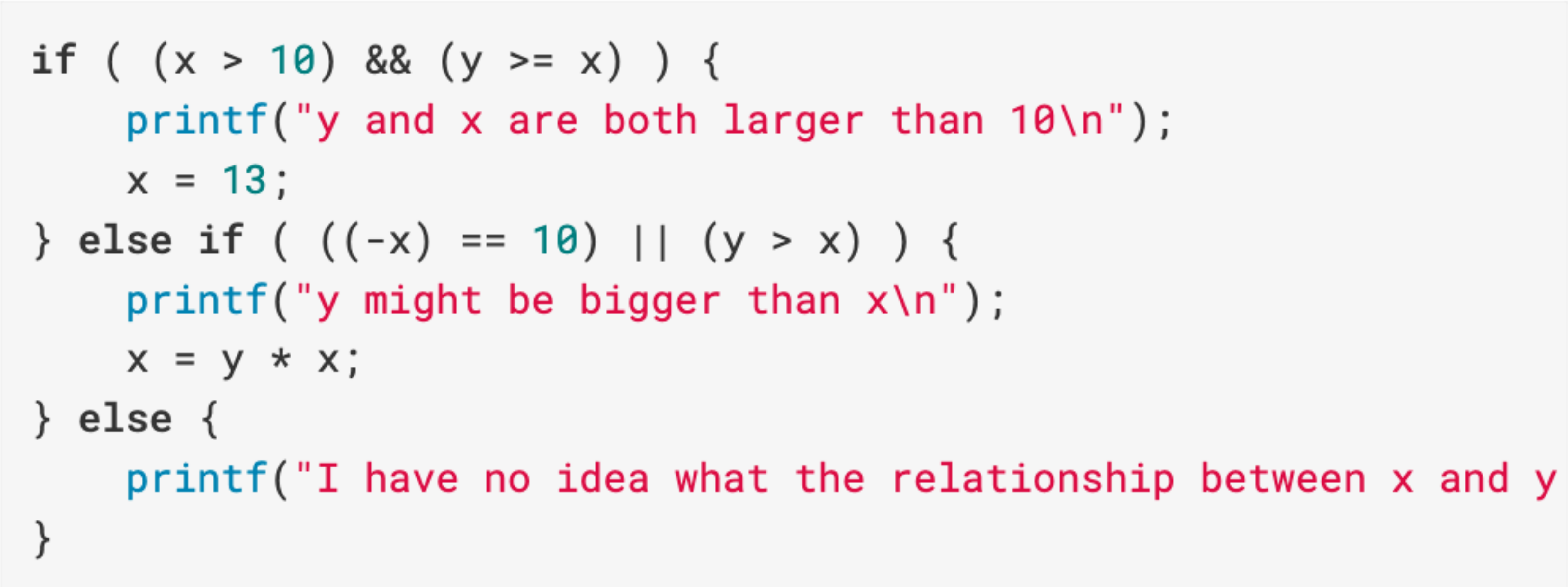
```
// assume x and y are ints, and have been assigned
// values before this point in the code

if (y < 0) {
    printf("y is negative\n");
} else if (y != 0) {
    printf("y is positive\n");
} else {
    printf("y is zero\n");
}

// set x and y to the larger of the two values
if (x >= y) {
    y = x;
} else {
    x = y;
}
```

Logical Operators

! logical negation

&& logical and &&

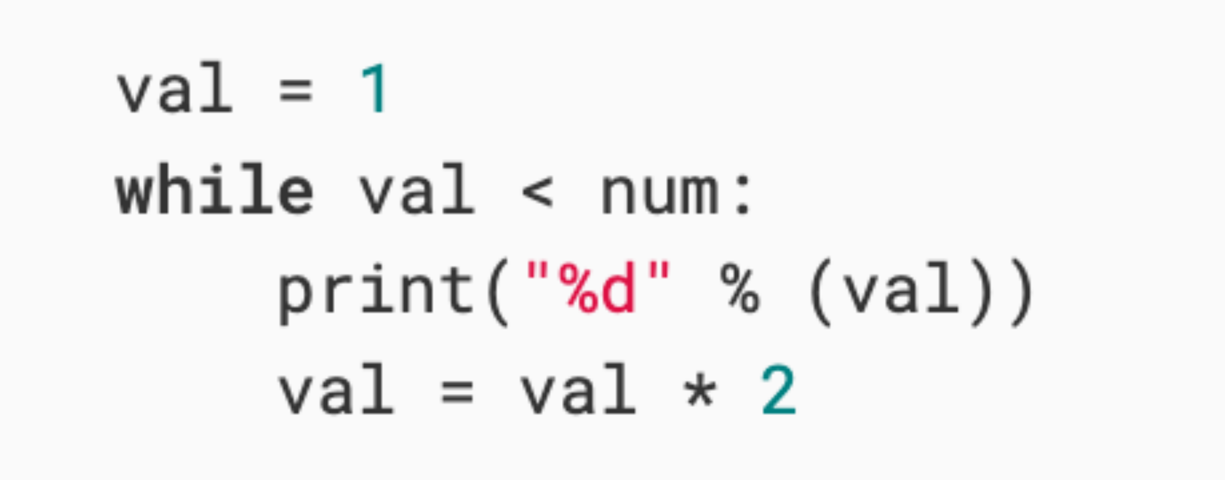
|| logical or ||



```
if ( (x > 10) && (y >= x) ) {
    printf("y and x are both larger than 10\n");
    x = 13;
} else if ( ((-x) == 10) || (y > x) ) {
    printf("y might be bigger than x\n");
    x = y * x;
} else {
    printf("I have no idea what the relationship between x and y
}
```

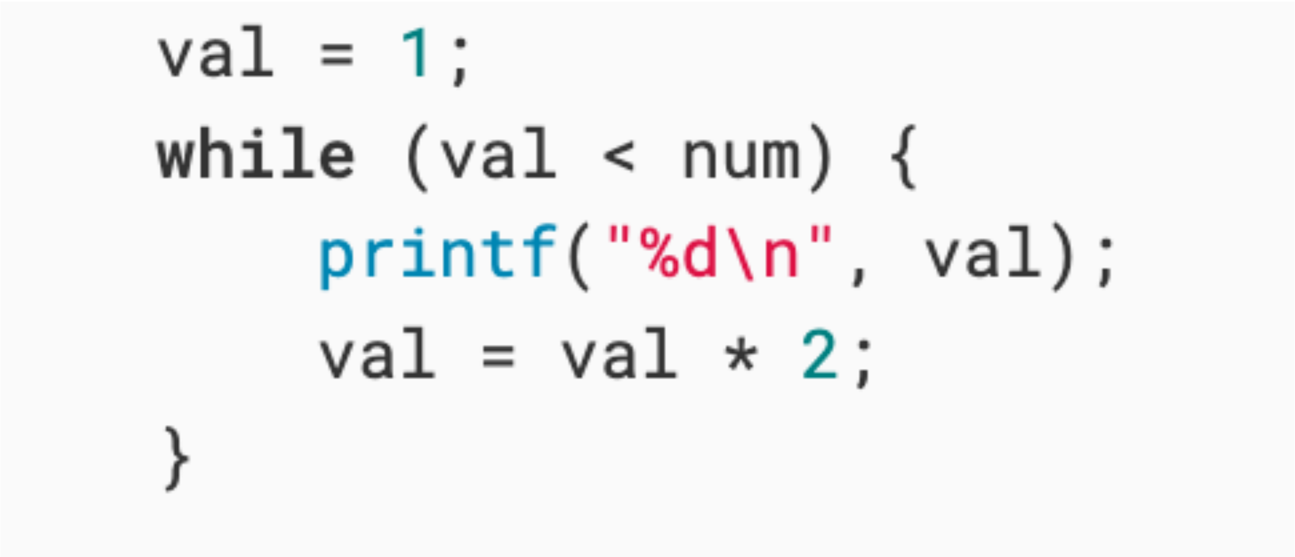
while Loops

Python

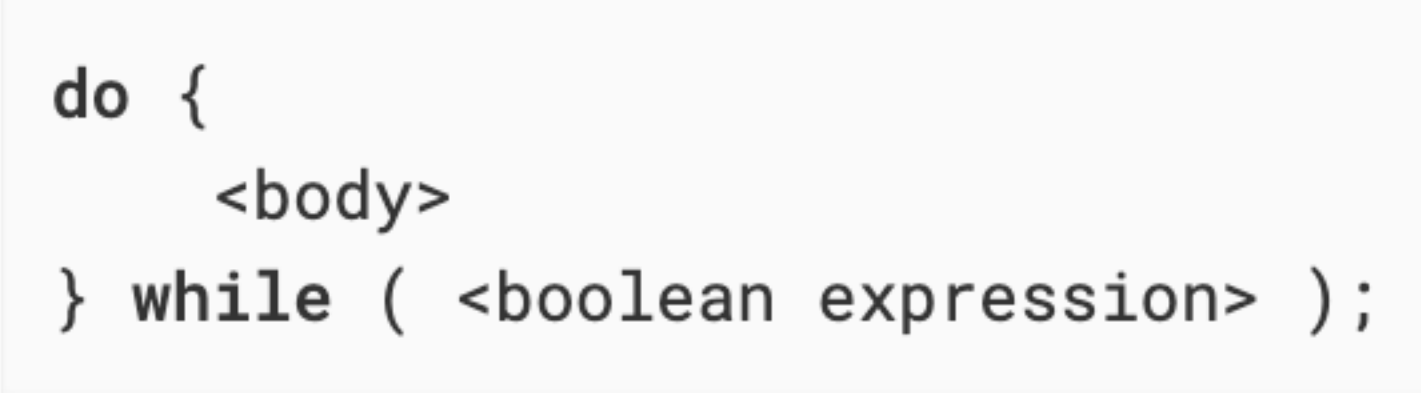


```python
val = 1
while val < num:
    print("%d" % (val))
    val = val * 2
```

C



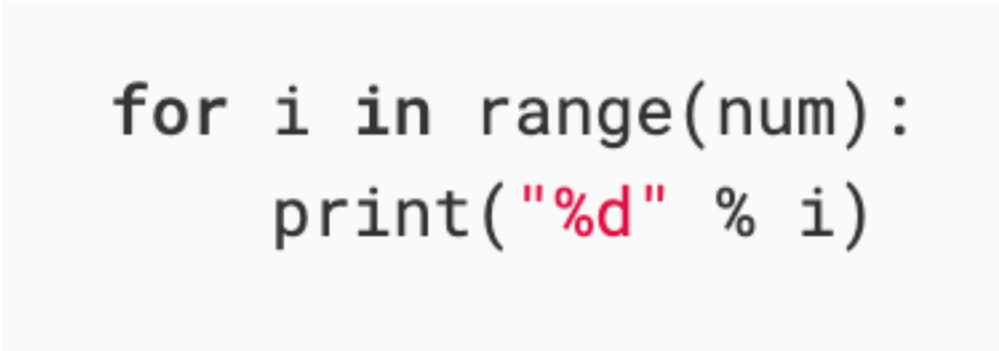
```c
val = 1;
while (val < num) {
    printf("%d\n", val);
    val = val * 2;
}
```



```c
do {
    <body>
} while ( <boolean expression> );
```

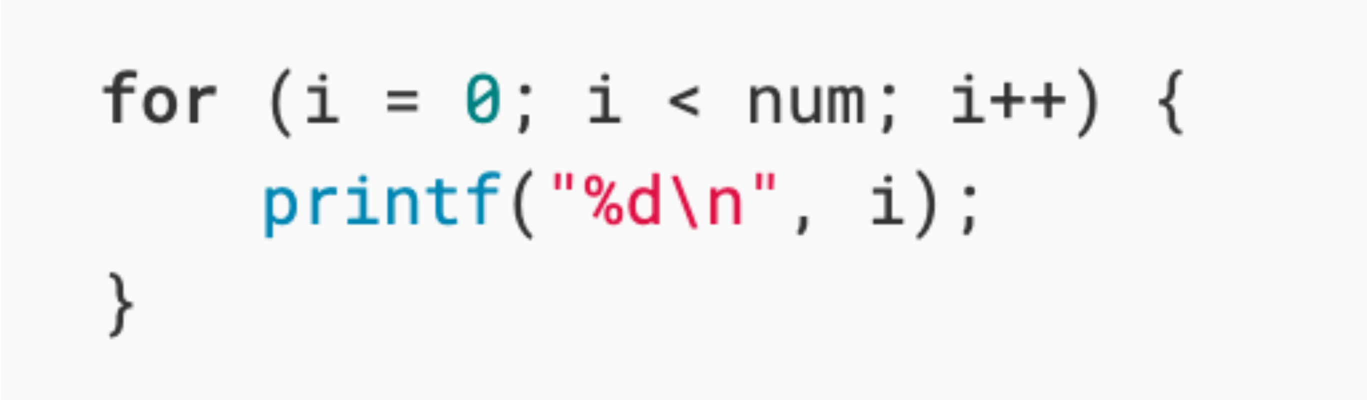
for Loops

Python



```python
for i in range(num):
    print("%d" % i)
```

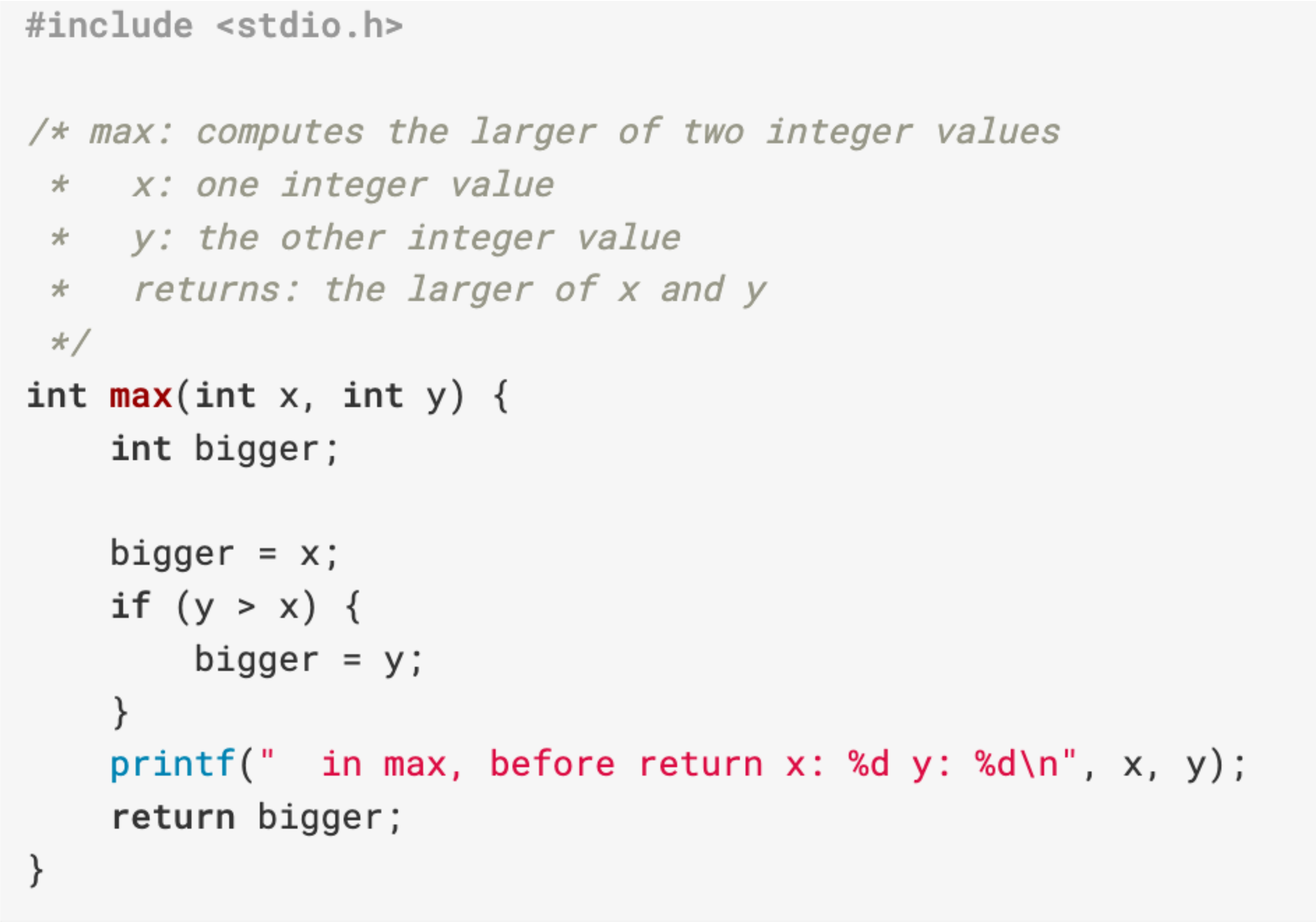
C



```c
for (i = 0; i < num; i++) {
    printf("%d\n", i);
}
```

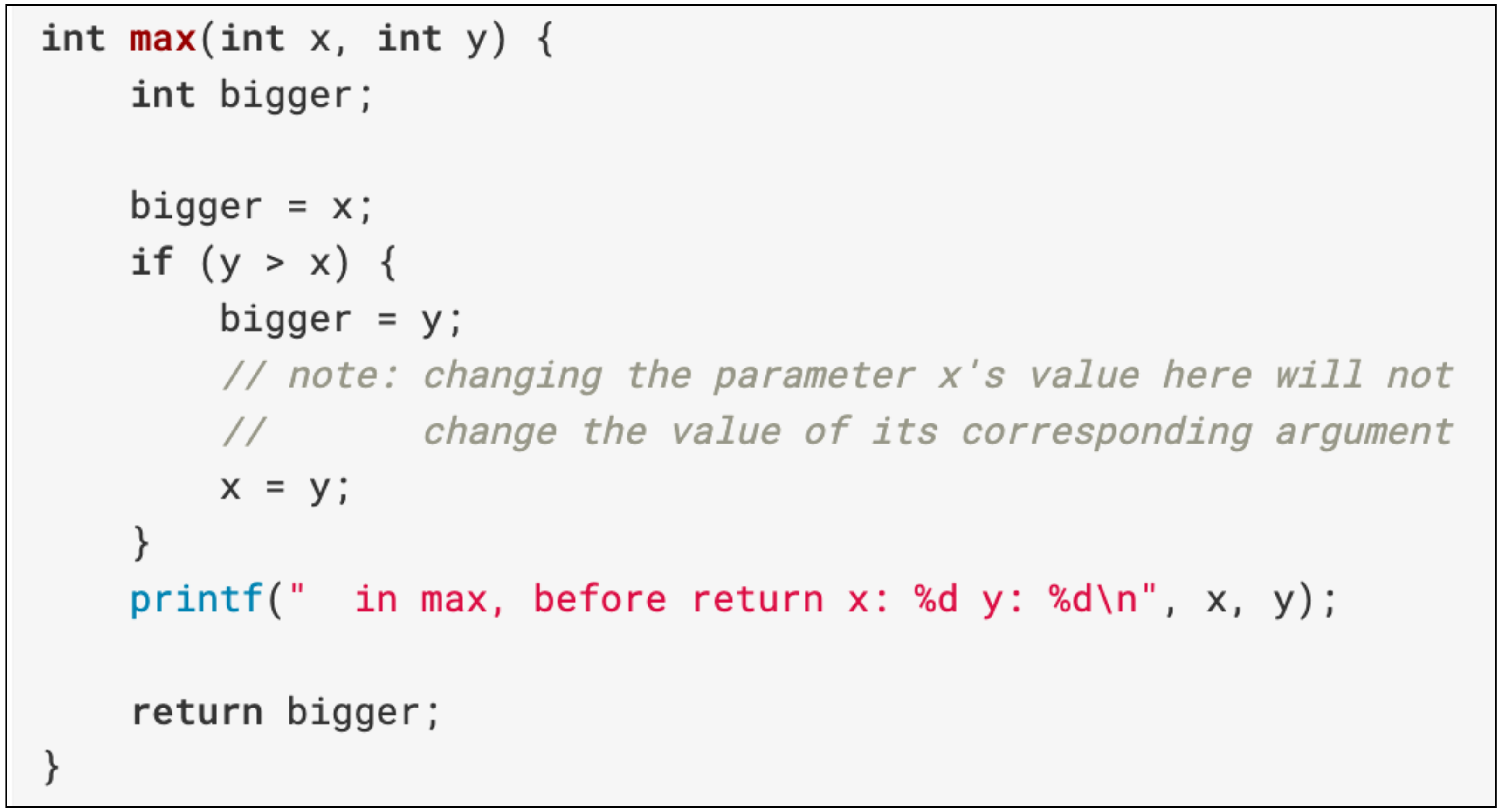
1.4. Functions

Function Example

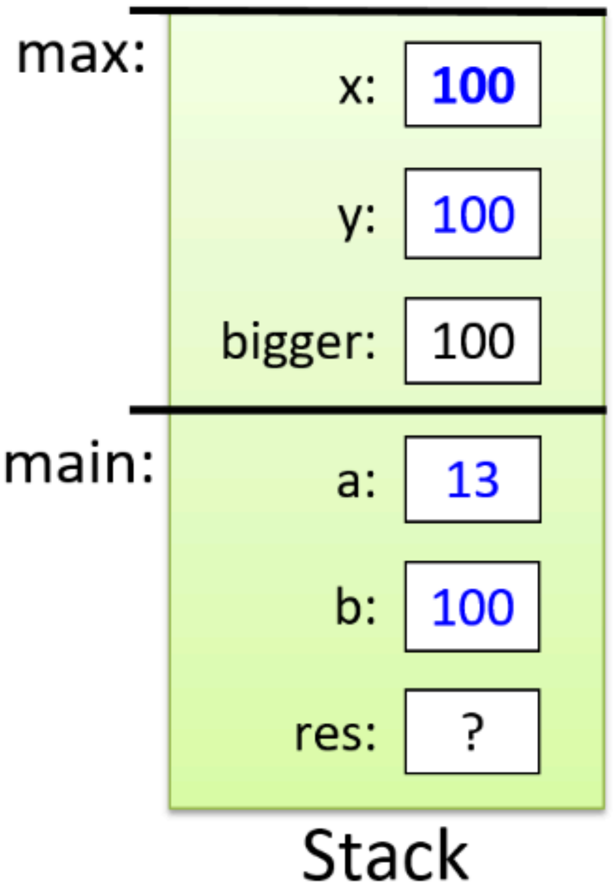


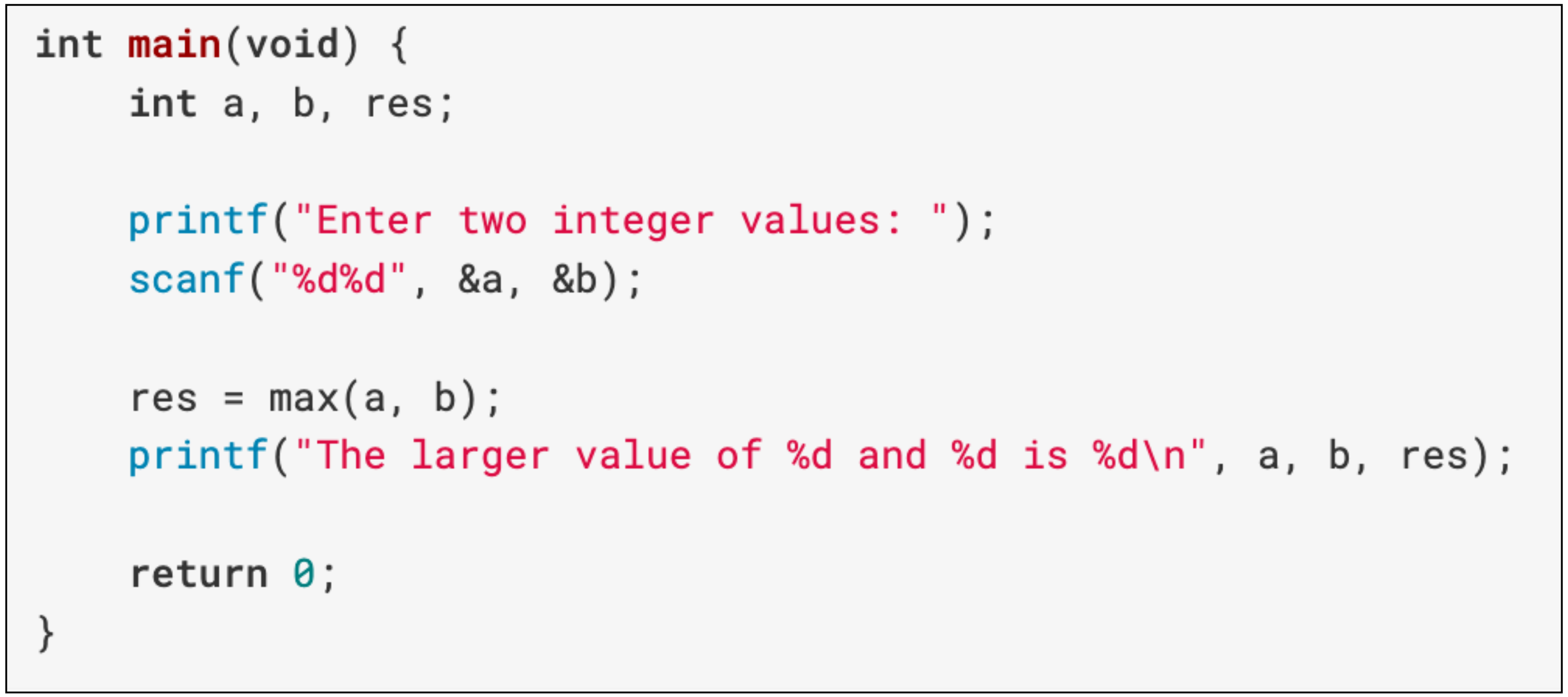
```c
#include <stdio.h>

/* max: computes the larger of two integer values
 *   x: one integer value
 *   y: the other integer value
 *   returns: the larger of x and y
 */
int max(int x, int y) {
    int bigger;

    bigger = x;
    if (y > x) {
        bigger = y;
    }
    printf("  in max, before return x: %d y: %d\n", x, y);
    return bigger;
}
```

1.4.1 The Stack



```c
int max(int x, int y) {
    int bigger;

    bigger = x;
    if (y > x) {
        bigger = y;
        // note: changing the parameter x's value here will not
        //       change the value of its corresponding argument
        x = y;
    }
    printf("  in max, before return x: %d y: %d\n", x, y);

    return bigger;
}
```



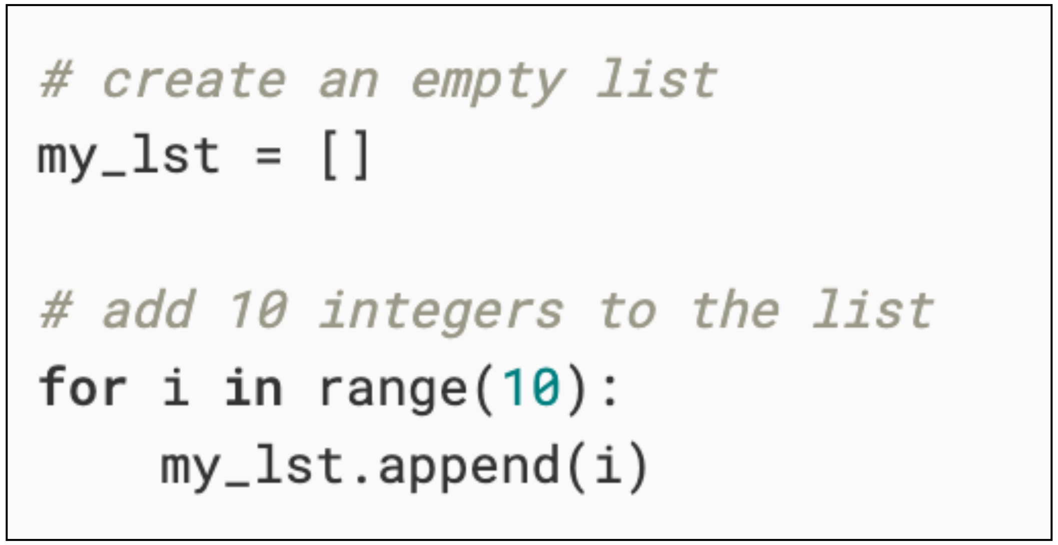


```c
int main(void) {
    int a, b, res;

    printf("Enter two integer values: ");
    scanf("%d%d", &a, &b);

    res = max(a, b);
    printf("The larger value of %d and %d is %d\n", a, b, res);

    return 0;
}
```

1.5. Arrays and Strings

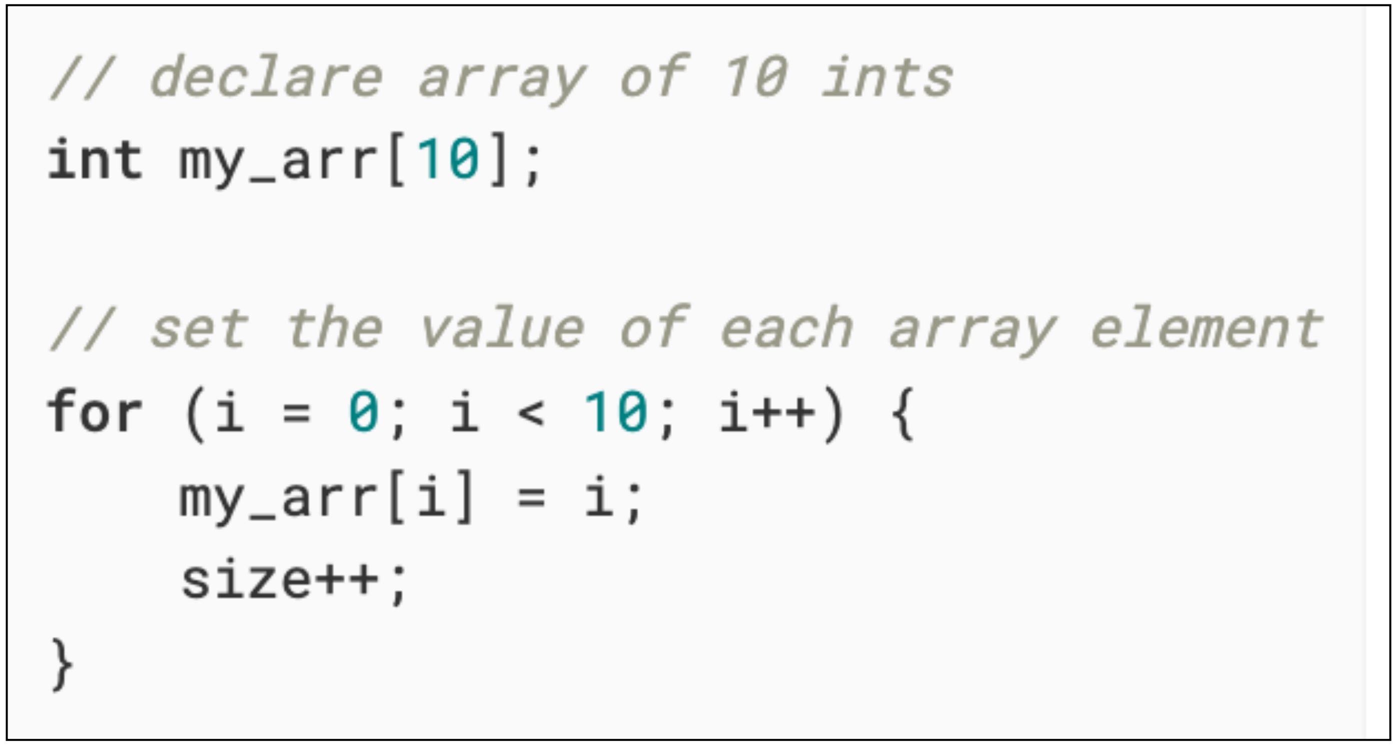
Arrays v. Lists

Python



```python
# create an empty list
my_lst = []

# add 10 integers to the list
for i in range(10):
    my_lst.append(i)
```

C

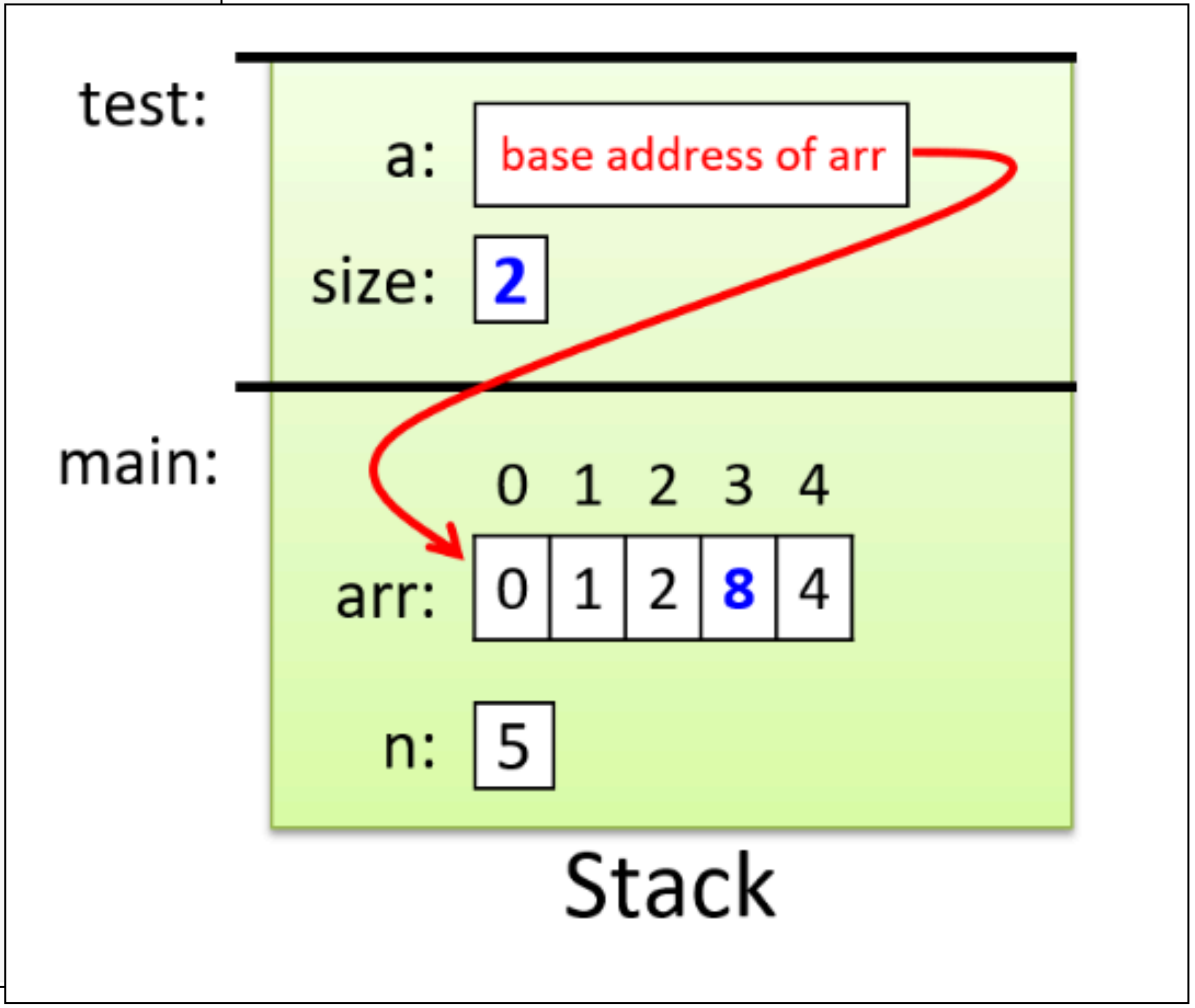


```c
// declare array of 10 ints
int my_arr[10];

// set the value of each array element
for (i = 0; i < 10; i++) {
    my_arr[i] = i;
    size++;
}
```

- A Python list can contain different types of data, **and resizes as needed**
- A C array's elements must all be the same size, and the **size does not change**
    - This leads to **buffer overflow errors**

Arrays and Functions

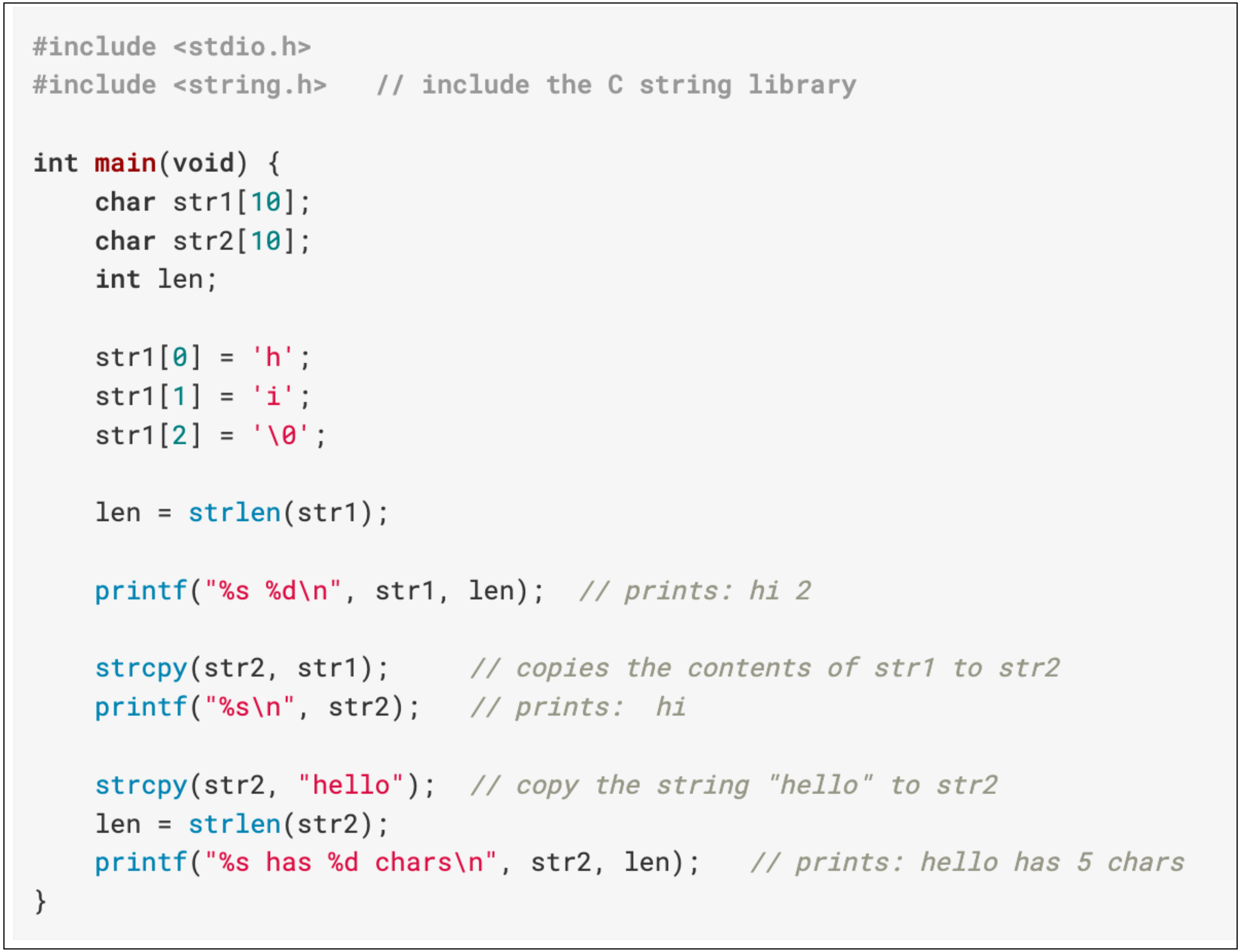
```
void test(int a[], int size) {
    if (size > 3) {
        a[3] = 8;
    }
    size = 2; // changing parameter does NOT change argument
}

int main(void) {
    int arr[5], n = 5, i;

    for (i = 0; i < n; i++) {
        arr[i] = i;
    }

    printf("%d %d", arr[3], n);  // prints: 3 5

    test(arr, n);
    printf("%d %d", arr[3], n);  // prints: 8 5

    return 0;
}
```



1.5.4 Strings and the C String Library

- Strings must end with a null byte **'\0'**

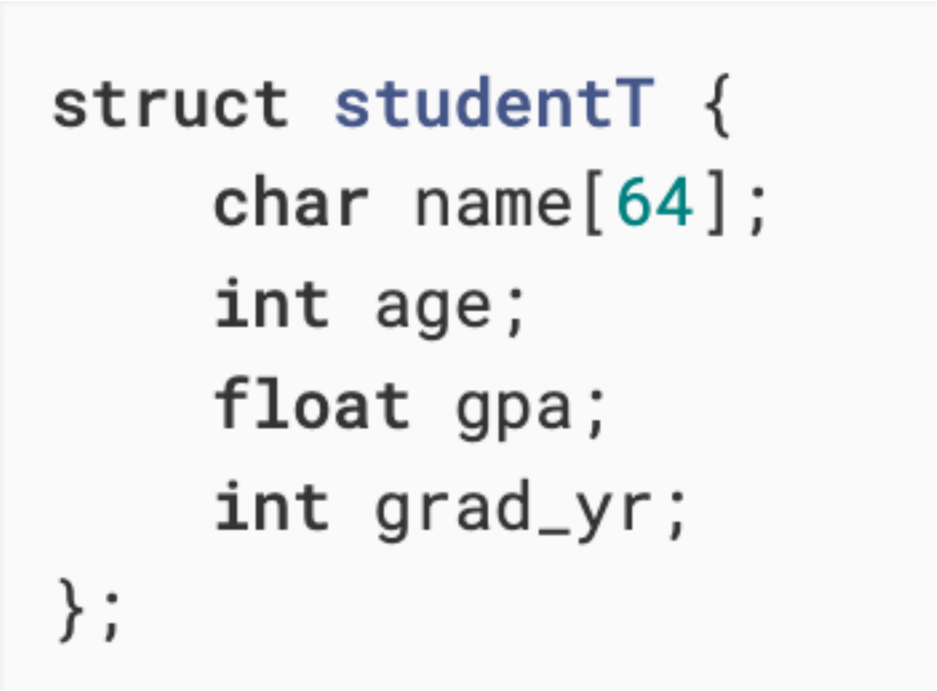
1.5.4 Strings and the C String Library



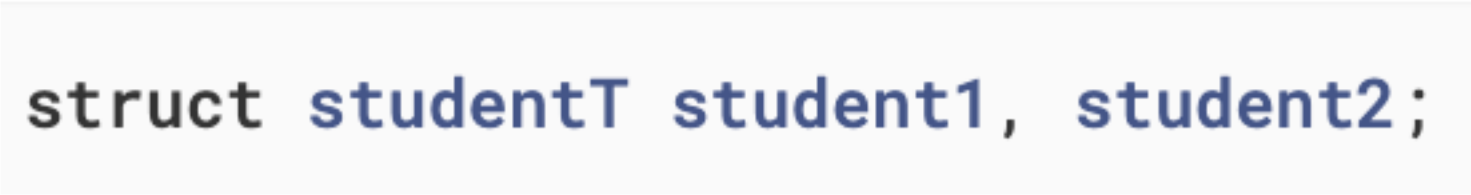
```c
#include <stdio.h>
#include <string.h>   // include the C string library

int main(void) {
    char str1[10];
    char str2[10];
    int len;

    str1[0] = 'h';
    str1[1] = 'i';
    str1[2] = '\0';

    len = strlen(str1);

    printf("%s %d\n", str1, len);  // prints: hi 2

    strcpy(str2, str1);     // copies the contents of str1 to str2
    printf("%s\n", str2);   // prints:  hi

    strcpy(str2, "hello");  // copy the string "hello" to str2
    len = strlen(str2);
    printf("%s has %d chars\n", str2, len);   // prints: hello has 5 chars
}
```

1.6. Structs

1.6 Structs



```
struct studentT {
    char name[64];
    int age;
    float gpa;
    int grad_yr;
};
```



```
struct studentT student1, student2;
```

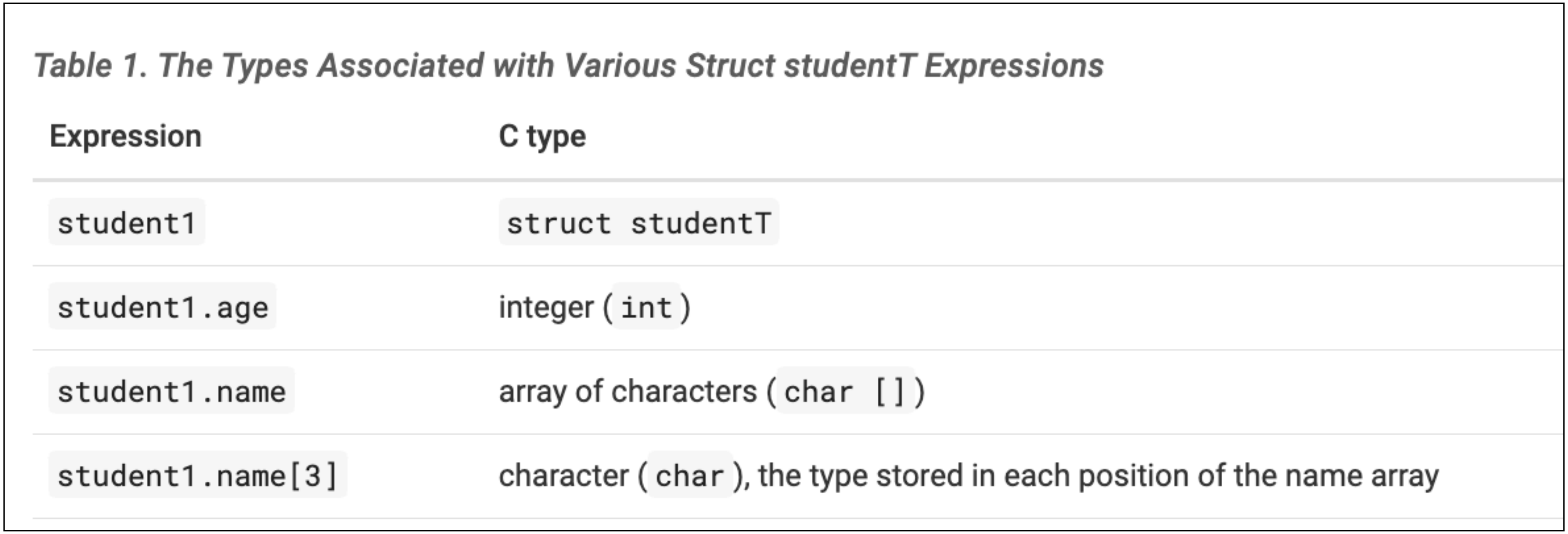


Table 1. The Types Associated with Various Struct studentT Expressions

Expression	C type
`student1`	`struct studentT`
`student1.age`	integer (`int`)
`student1.name`	array of characters (`char []`)
`student1.name[3]`	character (`char`), the type stored in each position of the name array

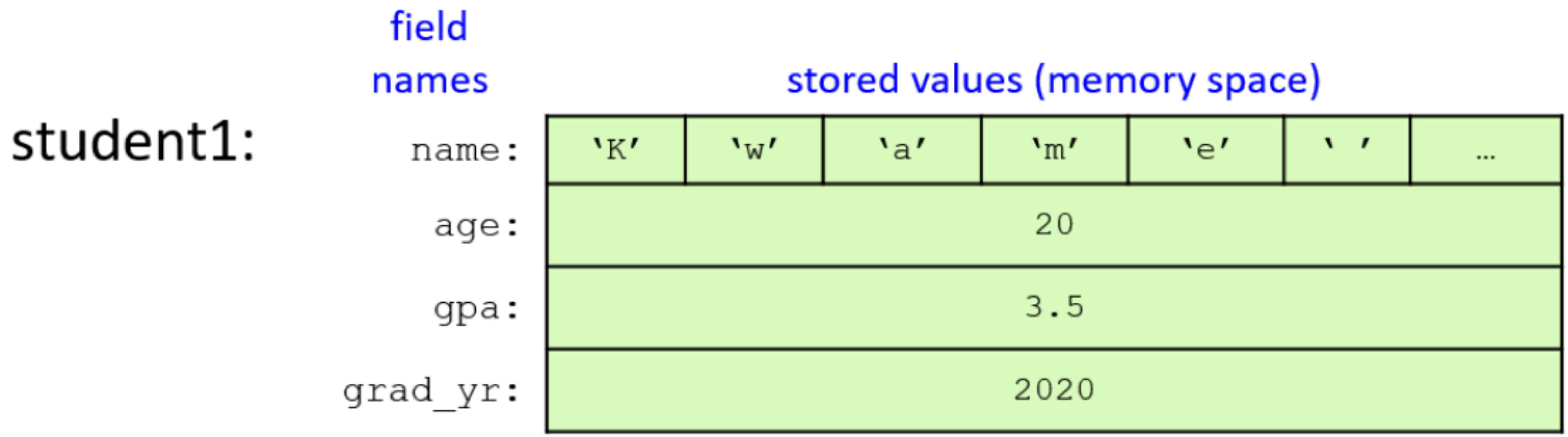


Figure 1. The student1 variable's memory after assigning each of its fields

Ch 1b